NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN OF A PIPELINED MULTIPLIER
USING A SILICON COMPILER

by

Ronald S. Huber

June 1990

Thesis Advisor: Herschel H. Loomis, Jr.

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report
2b Declassification/Downgrading Schedule	Approved for public release; distribution is unlimited.
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization Naval Postgraduate School — 6b Office Symbol (If Applicable) 39	7a Name of Monitoring Organization Naval Postgraduate School
6c Address (city, state, and ZIP code) Monterey, CA 93943-5000	7b Address (city, state, and ZIP code) Monterey, CA 93943-5000
8a Name of Funding/Sponsoring Organization — 8b Office Symbol (If Applicable)	9 Procurement Instrument Identification Number
8c Address (city, state, and ZIP code)	10 Source of Funding Numbers: Program Element Number / Project No / Task No / Work Unit Accession No

11 Title *(Include Security Classification)* Design of a Pipelined Multiplier Using a Silicon Compiler.

12 Personal Author(s) Ronald Scott Huber

13a Type of Report	13b Time Covered	14 Date of Report *(year, month,day)*	15 Page Count
Master's Thesis	From To	June 1990	

16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms *(continue on reverse if necessary and identify by block number)*
Field	Group	Subgroup	silicon compiler, pipeline, digital multiplier

19 Abstract *(continue on reverse if necessary and identify by block number:*

This thesis describes the design methodology and the process of employing the GENESIL Silicon Compiler (GSC) (Version 7.1) in the layout of a pipelined multiplier, in 1.5 micron CMOS technology, using a parallel multiplier cell array. Additionally, background material on the theory of multiplication, as well as the concept and theory of pipelining are presented.

The results revealed two practical limits of the GSC system which precluded achieving the high component density made possible by full custom, "manual" CAD methods using graphic layout tools. Although the GSC system did not perform as desired in this study, it offers a viable alternative to the labor intensive, full custom, VLSI graphic layout tools in use today.

20 Distribution/Availability of Abstract [X] unclassified/unlimited [ ] same as report [ ] DTIC users	21 Abstract Security Classification Unclassified	
22a Name of Responsible Individual Herschel H. Loomis, Jr.	22b Telephone *(Include Area code)* (408) 646-3214	22c Office Symbol EC/LM

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted — security classification of this page

All other editions are obsolete — Unclassified

Approved for public release; distribution is unlimited.

DESIGN OF A PIPELINED MULTIPLIER USING A SILICON COMPILER

by

Ronald Scott Huber
Lieutenant Commander, United States Navy
B.S., University of California at Riverside, 1976

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

June 1990

Author: Ronald Scott Huber

Ronald Scott Huber

Approved By:

Herschel H. Loomis, Jr. Thesis Advisor

Chyan Yang, Co-Advisor

John P. Powers, Chairman, Department of Electrical and
Computer Engineering

ABSTRACT

This thesis describes the design methodology and the process of employing the GENESIL Silicon Compiler (GSC) (Version 7.1) in the layout of a pipelined multiplier, in 1.5 micron CMOS technology, using a parallel multiplier cell array. Additionally, background material on the GSC, the theory of multiplication, as well as the concept and theory of pipelining are presented.

The results revealed two practical limits of the GSC system which precluded achieving the high component density made possible by full custom, "manual" CAD methods using graphic layout tools. Although the GSC system did not perform as desired in this study, it offers a viable alternative to the labor-intensive, full custom, VLSI graphic layout tools in use today.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

I would like to extend a heartfelt thank you to the following individuals for their time, patience, and energy in helping make this thesis come to fruition:

Prof. Herschel H. Loomis, Jr., Naval Postgraduate School

Prof. Chyan Yang, Naval Postgraduate School

LT Dave Stuart, USN, Class of June 1990, Naval Postgraduate School

And most importantly, a very special thank you to my wife Elena for her love and support which allowed me to successfully complete my graduate education, and for her endearing reminder during trying times that "there are always cans on the beach !"

I. INTRODUCTION

A. BACKGROUND

Multiplication is often an essential function in many digital systems. For example, a multiplier is a necessary part of any digital signal processing circuit [Ref. 1]. In many signal processing operations, such as correlation, convolution, filtering, and frequency analysis, one needs to perform multiplication [Ref. 2], and, in order to perform real-time signal processing, a high-speed multiplier is required [Ref. 3]. Additionally, in the majority of digital signal processing applications the critical processing paths usually involve many multiplications [Ref. 4]. Clearly, fast digital multipliers are one of the most important building blocks in Very Large Scale Integration (VLSI) chips for advanced digital signal processing.

In high-performance systems, many of the above operations are implemented with bipolar device technology, which consumes a significant amount of direct current (DC) power. On the other hand, Complementary Metal Oxide Semiconductor (CMOS) technology can substantially reduce the power consumption, but results in much slower device speed.

CMOS is a combination of P-channel and N-channel enhancement metal oxide semiconductor field effect transistors (MOSFETs) used in a complementary circuit arrangement that is useful in digital logic circuitry. Among its advantages are that it has extremely low power dissipation, requires only one DC power supply, operates over a wide range of supply voltages, and can drive as many as 50 gate-inputs [Ref. 5]. The fabrication of a CMOS IC (integrated circuit) requires a "prescription" for preparing the photomasks that will be used in the manufacturing process. This "prescription" is a set of rules which provides a link between the circuit designer and process engineer during the manufacturing phase. The rules are often referred to as layout rules or as design rules. The main objective of the layout rules is to make a circuit with optimum yield in as small an area (geometry) as possible without jeopardizing the reliability of the circuit [Ref. 2]. There are several ways to describe the design rules. One way is by the "micron" rules which are stated as some micron resolution. Micron design rules are usually given as a list of minimum feature sizes and spacing required for all the masks in a given fabrication process [Ref. 2]. Hence, as indicated in the abstract of this report, the multipliers designed in this thesis have a minimum feature size of 1.5 microns in CMOS technology. By incorporating pipelining into the design, the throughput of a large CMOS circuit can be improved significantly [Ref. 4]. For example, the results of a study by Hallin and Flynn [Ref. 6] indicated that pipelining can give a 40 percent increase in adder efficiency and a 230 percent increase in multiplier throughput.

With the advent of high-speed semiconductor memory, an increasing mismatch between memory access and multiplication time has arisen. Consequently, there is considerable interest in parallel array multipliers [Ref. 7]. An array multiplier and a multiplier using a Wallace tree are well-known for their high-speed multiplication [Ref. 3]. The previous study by Hallin and Flynn [Ref. 6] also demonstrated that the most efficient multiplier is a maximally pipelined tree multiplier which was shown to be 50 percent more efficient than the array multiplier. However, because unit cells in the array multiplier are used repeatedly its layout is highly modular. Modularity makes the array multiplier more favorable than a tree multiplier for VLSI implementation. Therefore, many MOS multipliers have been fabricated using this method [Ref. 3].

As ICs grow increasingly more complex, it becomes necessary to develop new methods to manage the design complexities, as well as the expenses associated with the design and testing of the IC. Also, from this increase in IC complexity arises the demand for faster and more economical methods to streamline the design process. One state-of-the-art solution to meet this demand is the silicon compiler. A silicon compiler is a computer system which generates IC layouts from high-level descriptions. The advantage that a silicon-compiler-based process has over a custom IC system design process is that the latter requires a team of experts in the fields of logic implementation, circuit simulation, chip layout, and testing. However, the design process based on the silicon compiler may be accomplished by one individual utilizing a top-down, hierarchical design methodology beginning with a partitioned chip set, progressing downward into individual chips and modules, and terminating at the block level. There is far less time required to design a IC using a silicon compiler than for a full custom, "manual" CAD method using graphic layout tools. Thus, one can see that the silicon compiler provides a streamlined method for rapid development of IC systems [Ref. 8]. The disadvantages of the silicon compiler are that the resulting circuit is often slower and the layout is not always efficient in its use of area.

B. THESIS GOALS

The motivation for this thesis was to learn more about digital multipliers, as well as to work with state-of-the-art VLSI circuit design tools. The main goal of this thesis was to design a pipelined multiplier using the GENESIL Silicon Compiler. Concomitant with this goal was the desire to learn more about the concept and theory of pipelining. An emphasis has been placed on documenting the thought processes that went into the multiplier designs in this thesis, as well as the problems encountered along the way. Additionally, it was a goal to fully explore and probe the GENESIL Silicon Compiler to determine its practical limits in parallel multiplier array design. Finally, there was an attempt to produce a document that could be understood by one not well versed in digital design methodology by first reviewing the basis concepts of digital multipliers and then discussing the concept and theory of pipelining.

The following is a description of each of the chapters which follow:

Chapter 2: Introduces the reader to the GENESIL Silicon Compiler.

Chapter 3: Presents three multiplier formats: serial, serial/parallel, and parallel.

Chapter 4: Presents the basic concepts of pipelining.

Chapter 5: Discusses the design process of a pipelined multiplier array.

Chapter 6: Discusses the limitations of the silicon compiler.

Chapter 7: Concludes the thesis with a summary and recommendations for follow on multiplier design.

II. GENESIL SILICON COMPILER

A. INTRODUCTION

The purpose of this chapter is to introduce the reader to the GENESIL Silicon Compiler (GSC) system. The intent is to present a broad overview of GSC capabilities so that the reader may become acquainted with the features used in this report. For a detailed description of the GSC system the reader is referred to References 9 through 11.

B. GENESIL SYSTEM DESCRIPTION

The GSC system is a design automation software system which allows systems engineers and circuit designers to design complex VLSI computer chips. GENESIL produces IC designs from architectural descriptions and allows for their verification. Figure 1 shows a block diagram of the GSC development system and Figure 2 depicts the overall layout of the GSC system hardware. The GSC design tasks and activities are listed in Figure 3 and it is these activities that will be emphasized in this chapter.

The GSC is based on an object-oriented hierarchical system running under the UNIX operating system. The objects consist of Blocks, Modules, Chips, and Chip-sets.

Use of the GSC system does not require design considerations at the transistor gate level. A systems engineer or circuit designer can simply incorporate into his layout one of the myriad of GSC circuits resident in the GSC library. The resident circuits in the GSC library consist of random access memory (RAM), read only memory (ROM), programmable logic arrays (PLA), arithmetic logic units (ALU), multipliers, and several less complex circuits such as basic logic gates and data-path elements [Ref. 12].

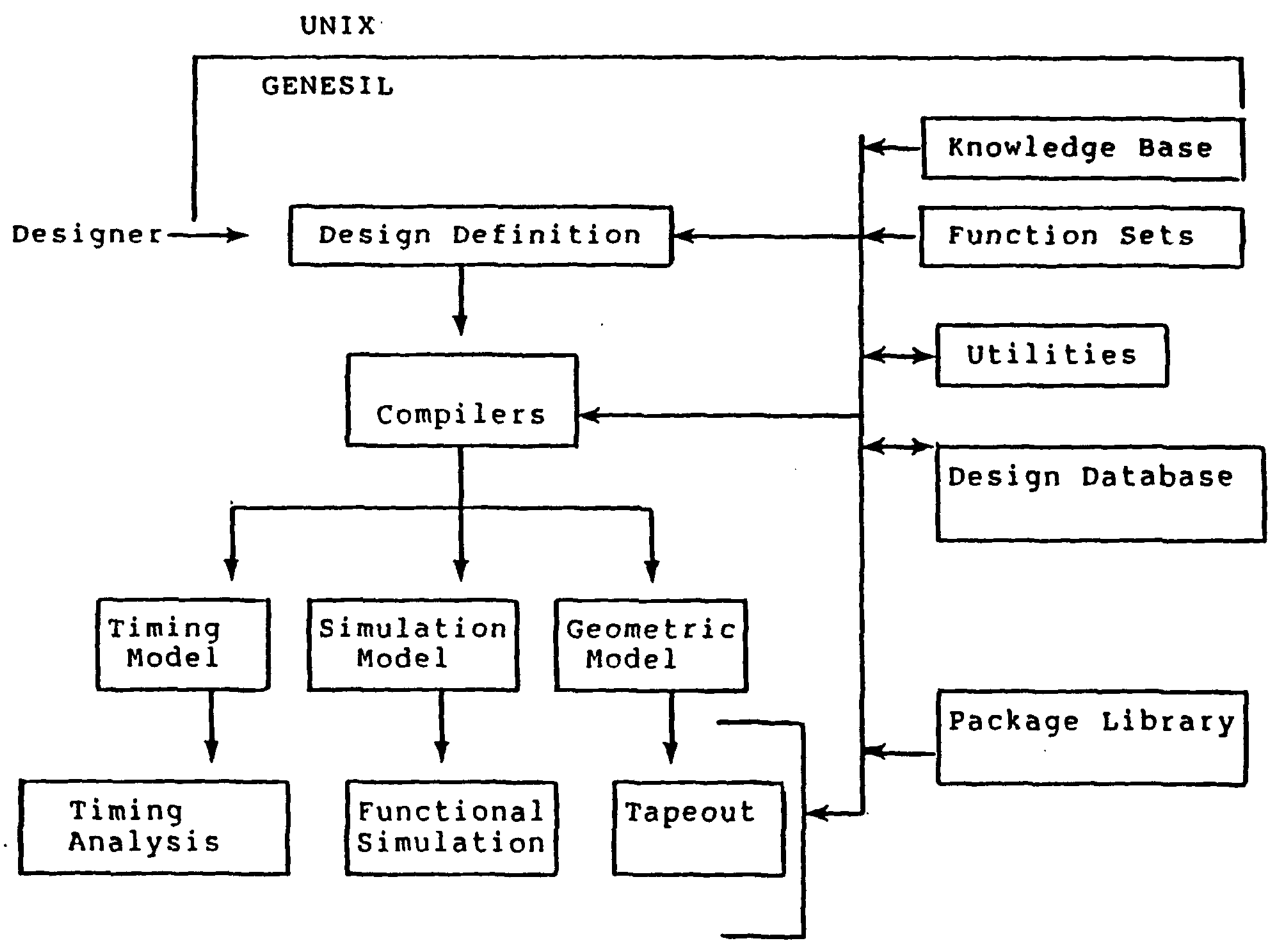


Figure 1 GENESIL Silicon Compiler Developmental System [From Ref. 9]

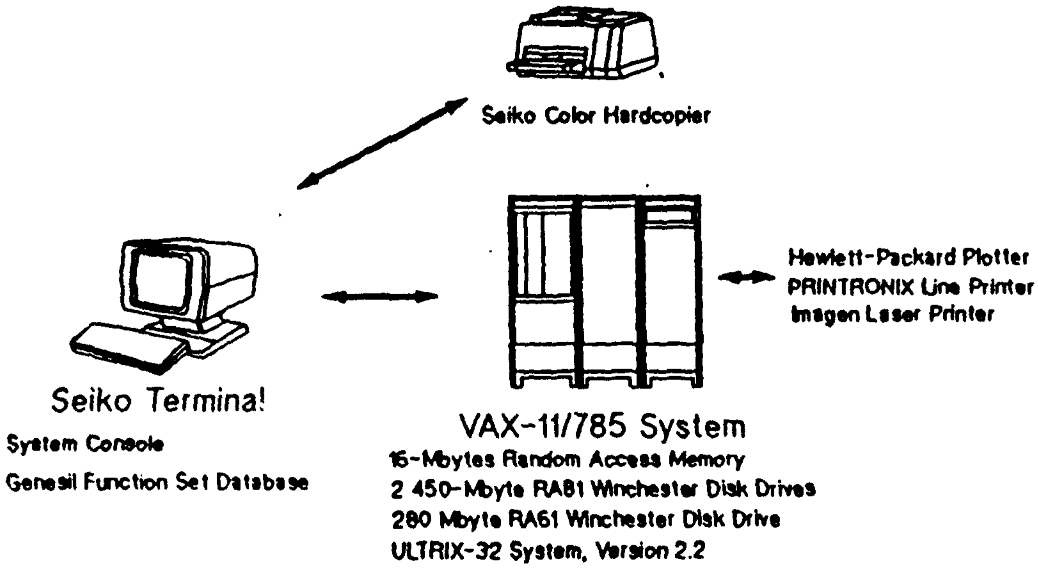


Figure 2 GENESIL Silicon Compiler Hardware System [From Ref. 12]

Tasks/Activities	*Menus/Commands	Forms
Design Description	*SELECT_OBJECT ATTACH_NEW	
	*SELECT_OBJECT UP DOWN PATH	
Definition	SPECIFICATION	Specification Form
	HEADER	Header Form
Netlisting	*NET_NETLIST	Net Netlist Form
	*OBJECT_NETLIST	Object Netlist Form
Floorplanning	*PLACEMENT	Placement Form
	*FUSION	Fusion Form
	*PINOUT	Pinout Form
Compiling	*COMPILE	
Functional Simulation	*SIMULATE	Environment Form Setup Form
Timing Analysis	*TIMING ANALYSIS	Timing Analysis Form
Manufacturing Interface	*PACKAGE_EDIT	
	*TAPEOUT	
	*PLOT	Plot Form

Figure 3 GENESIL Design Activities [From Ref. 12]

Before leaving this section the reader should become acquainted with the following tasks and activities of the GSC development system in order to derive the maximum benefit from the design process described in Chapter 5. For a detailed explanation of each task or activity the reader is referred to [Refs. 9-11].

C. TASKS AND ACTIVITIES

1. DEFINITION

The DEFINITION activity is the process whereby the user defines an object using the options provided in the DEFINITION menu. Defining an object consists of accessing the HEADER and SPECIFICATION forms from the DEFINITION menu.

A. HEADER

Use of the HEADER option allows the user to display the HEADER form, which is dependent on the current object connected to the user's account. The HEADER form allows the user to specify the technology and fabrication lines (fablines) to be utilized in the users design. The selected choice propagates down the entire hierarchy. The fabline selection process used in this thesis will be discussed in Chapter 5.

B. SPECIFICATION

Use of the SPECIFICATION form, which is also dependent upon the current object attached to the user's account, allows the user to fill in detailed object characteristics. For example, if one were using a FIFO Block in his design, he could specify its width, depth, output register, and connectors through use of the SPECIFICATION form.

2. NETLISTING

NETLISTING allows the user to specify the interconnections between Blocks and Modules to form higher level functional Modules. This is accomplished through the use of NET_NETLIST and OBJECT_NETLIST. It should be noted that they both provide the same information but from different points of reference.

A. NET_NETLIST

NET_NETLIST is used to specify the signal names to be connected into a network, and once they are defined, the GENESIL System then creates the network.

B. OBJECT_NETLIST

OBJECT_NETLIST allows the user to specify the signals on Blocks or Submodules in a Module or Modules in a Chip, and the GENESIL system then creates the connections between the specified objects.

The author found these two options to be the most important of the GSC options used in this thesis. A mastery of these two options is paramount to a successful and trouble-free design evolution. It was preferable to establish the initial connections with OBJECT_NETLIST, and, if errors arose, they were investigated with NET_NETLIST. NET_NETLIST allows one to trace signal names and their associated connections.

3. FLOORPLANNING

FLOORPLANNING is the placement of objects on the Chip, the specification of their FUSION order, and the connection of the pins to the pads of the Chip. The FLOORPLANNING task prepares the design objects for routing. One should be aware the FLOORPLANNING activities have a significant influence on the efficiency of the router. FLOORPLANNING consists of the following activities:

A. *PLACEMENT*

PLACEMENT specifies an object's location relative to other objects in a Module or Chip. This is usually done graphically by either selecting the GSC AUTO-PLACEMENT option or by manual PLACEMENT by the user. In almost all cases the author preferred manual PLACEMENT over AUTO-PLACEMENT. A further discussion of the PLACEMENT activity will be held in Chapter 5.

B. *FUSION*

The FUSION activity allows the user to graphically create and modify the assignments of routing channels on the floorplan to influence wire routing. This option was not frequently used in this study although some experimentation was conducted. There was no real enhancement observed to the designs in this thesis when employing this option. Because the compiling process and the plotting of the layout designs were very time-consuming (on the order of several hours for large layouts), it was difficult to the justify the investment of time for what little effect (if any) was observed.

C. *PINOUT*

PINOUT assigns external signals, both on and off the Chip. The user must be aware of the assignment of pins as it affects the routing both on and off the Chip.

4. COMPILE

The COMPILE activity can be initiated by the user or by the GENESIL system. GENESIL automatically performs a currency check on all objects, and if any are determined to be out of date it does a compile before any of the activities requiring compilation. A design must first be compiled before any significant activity can be started. Here, the author found it to be a time-saving investment if modular subcomponents were first compiled prior to building larger arrays incorporating these same subcomponents.

5. FUNCTIONAL SIMULATION

A. SIMULATE

SIMULATE is the operation to simulate the logical functioning of the IC design under consideration. One may test the IC design using automatic test vectors or by initiating manual simulation by *binding* the input pins to a "0" or "1" and manually advancing the time. Note that this process does not check the timing of the circuit. The manual method was used to test and simulate the designs reported on in this thesis. For large numbers, the product was verified with an HP-28S hand-held calculator. This topic is elaborated on in Chapter 5.

6. TIMING ANALYSIS

The GENESIL Timing Analyzer can calculate and report on the following areas:

- Speed at which the object under analysis will run.
- Paths that limit the clock frequency.
- Duty-cycle (phase high time) constraints.
- Input setup and hold times.
- Output delays.
- Setup and hold times and signal delays for any internal nodes.
- Path delays between internal nodes.

III. MULTIPLIER BASICS

A. BASIC MULTIPLIER DESIGN

This section provides a brief review of basic multiplier design as background before discussing the parallel multiplier arrays implemented in this report. The formats that will be discussed are the serial form, serial/parallel form, and the parallel form; the Wallace tree multiplier will also be briefly discussed. One should keep in mind that the selection of a specific multiplier to be incorporated in a particular design is based on speed, throughput, numerical accuracy, and area [Ref. 2].

Before beginning a discussion on the various forms mentioned above, the most basic form of multiplication will be discussed first. This is shown in Figure 4 which illustrates the multiplication of two positive binary integers, 14_{10} and 7_{10}.

$$
\begin{array}{lrcr}
\text{multiplicand;} & 1110 & : & 14_{10} \\
\text{multiplier ;} & \underline{0111} & : & 7_{10} \\
 & 1110 & & \\
 & 1110 & & \\
 & 1110 & & \\
 & \underline{0000} & & \\
 & 1100010 & : & 98_{10}
\end{array}
$$

Figure 4   Basic Form of Multiplication

The multiplication is accomplished through successive additions and shifts. This multiplication process may be separated into the following two steps:

- Evaluation of partial products.
- Addition of the shifted partial products.

It should be pointed out that one-bit binary multiplication is equivalent to a logical AND operation. Thus, the evaluation of partial products consists of the logical ANDing of the multiplicand and its associated bit in the multiplier.

1. Serial Multiplier

The simplest example of a serial multiplier is illustrated in Figure 5. Here, multiplication is accomplished through a successive addition algorithm and is implemented using a full adder, a logical AND, a delay element, and a serial-to-parallel register. The numbers X and Y are presented serially to the circuit and the partial product is evaluated for each bit of the multiplier. Next, a serial addition is performed with the partial additions previously stored in the register. The G2 gate resets the partial sum at the beginning of the multiplication cycle [Ref. 2].

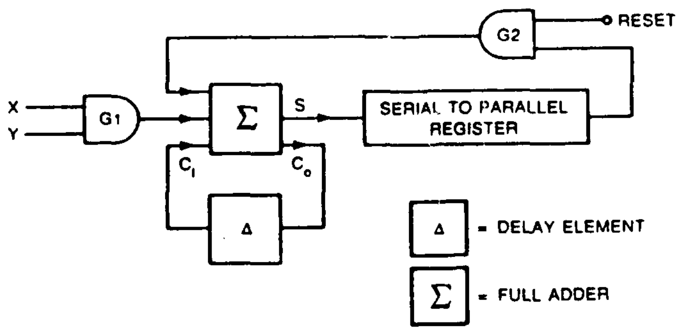


Figure 5 Basic Serial Multiplier [From Ref. 2]

2. Serial/Parallel Multiplier

The basic implementation of the serial/parallel multiplier form is illustrated in Figure 6. Here, multiplication is performed by successive additions of columns of the shifted partial products. As left-shifting by one bit in serial systems is accomplished by a 1-bit delay element, the multiplier is successively shifted and gates the appropriate bit of the multiplicand. The bits of the delayed, gated multiplicand must all be in the same column of the shifted partial product. They are added to form the product bit corresponding to the appropriate column [Ref. 2].

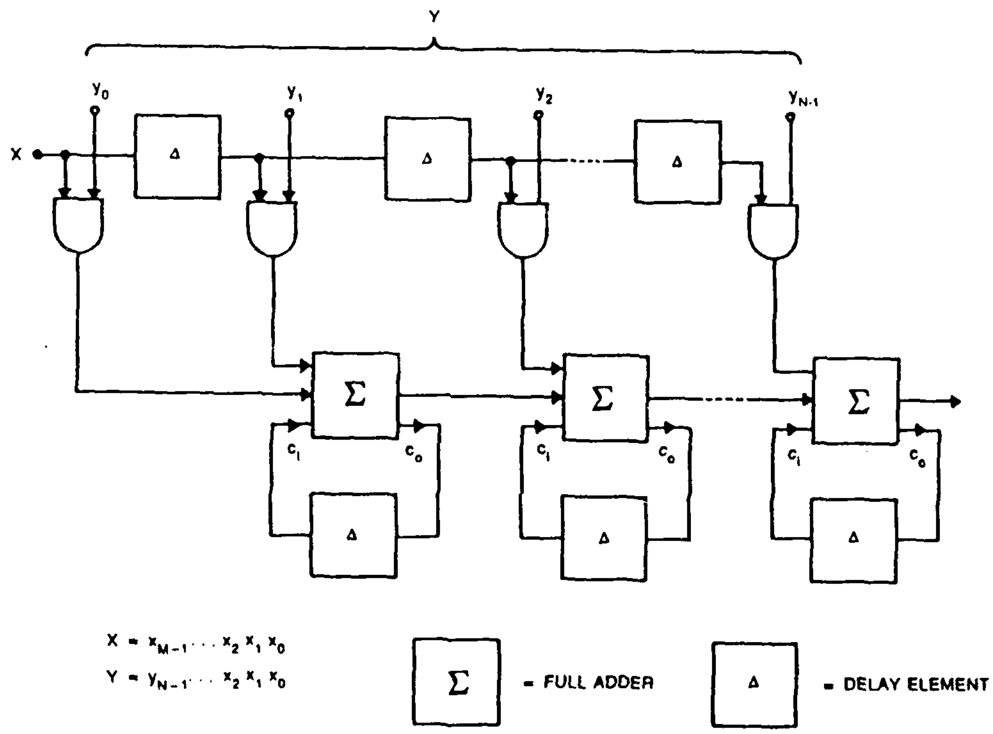


Figure 6 Basic Structure for Serial/Parallel Multiplier [From Ref. 2]

3. Parallel Multiplier

The parallel multiplier form is the one utilized in the design of the multipliers in this thesis. This form was selected primarily because, when incorporated into an array, the unit cells of the multiplier can be used repeatedly, resulting in a highly modular arrangement. Recall that this characteristic makes the parallel array multiplier favorable for VLSI implementation.

In a parallel multiplier the partial products in the multiplication process can be independently computed in parallel. For example, in the case of two unsigned binary integers X and Y:

$$X = \sum_{i=0}^{m-1} X_i 2^i \tag{3.1}$$

$$Y = \sum_{j=0}^{n-1} Y_j 2^j \tag{3.2}$$

The product is found by

$$P_r = X_y Y_r = \sum_{i=0}^{m-1} X_i 2^i \cdot \sum_{j=0}^{n-1} Y_j 2^j \tag{3.3}$$

$$= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (X_i Y_j) 2^{i+j}$$

The partial product terms P_k are called summands. There are *mn* summands, which are produced in parallel by the multiplication of *mn* AND gates [Ref. 2]. Figure 7 illustrates the partial products formed by the multiplication of two 4-bit numbers.

				X3	X2	X1	X0	Multiplicand
				Y3	Y2	Y1	Y0	Multiplier
				X3Y0	X2Y0	X1Y0	X0Y0	
			X3Y1	X2Y1	X1Y1	X0Y1		
		X3Y2	X2Y2	X1Y2	X0Y2			
	X3Y3	X2Y3	X1Y3	X0Y3				
P7	P6	P5	P4	P3	P2	P1	P0	Product

Figure 7 4-Bit Multiplier Partial Products [From Ref 2]

For an n x n multiplier the required number of components would be n(n-2) full adders, n half adders, and n^2 AND gates. The worst-case delay associated with such a multiplier is $(2n = 1)\tau_g$, where τ_g is the worst-case adder delay. Figure 8 illustrates a typical parallel multiplier cell which forms the basis of the multipliers designed in this thesis.

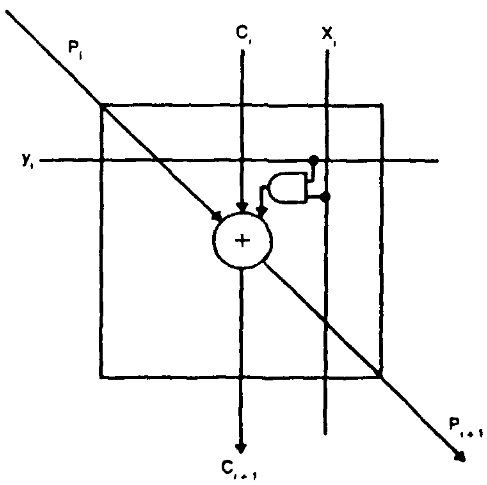


Figure 8 Parallel Multiplier Cell [From Ref. 2]

Note in Figure 8 above, that the X_i term is propagated vertically, while the Y_j term is propagated horizontally, and that the partial products enter at the top left of each cell. A bit-wise AND is performed in each cell, and the SUM (P_{i+1}) is forwarded to the next cell at the lower right. The CARRY OUT (C_{i+1}) is forwarded out the bottom of the cell. Figure 9 illustrates a parallel multiplier array with the partial products formed within each parallel multiplier cell.

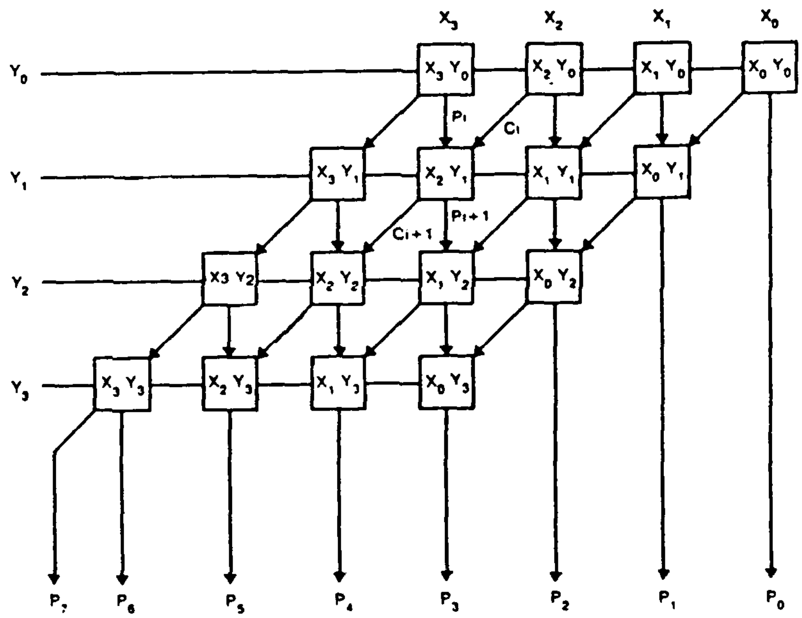


Figure 9 Parallel Multiplier Array [From Ref. 2]

As alluded to earlier, an important feature of the parallel multiplier array is that the unit cells of the multiplier can be used repeatedly, resulting in a highly modular arrangement. This arrangement of parallel multiplier cells can be drawn as a square array as indicated in Figure 10. Here, one can clearly see how the X_i and Y_i terms are propagated throughout the array by vertical and horizontal feedthrough, respectively. As mentioned previously, this feature makes the parallel array multiplier highly favorable for VLSI implementation.

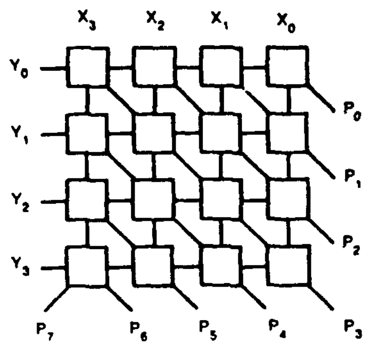


Figure 10 Parallel Multiplier Array Drawn as a Square Array [From Ref. 2]

4. Wallace Tree

A general discussion of digital multiplier design would not be complete without some mention of the Wallace tree. As stated earlier, a study by Hallin and Flynn [Ref. 6] demonstrated that the most efficient multiplier is a maximally pipelined tree multiplier which was shown to be 50 percent more efficient (with less overall delay) than an array multiplier.

The Wallace tree layout (Figure 11) is significant in that it utilizes a matrix generation and reduction scheme, which is the fastest way to perform parallel multiplication. However, it has some disadvantages when implemented in VLSI. The full Wallace tree is topologically difficult to implement. Large Wallace trees are difficult to map onto planes since each carry-save adder communicates with its own slice, transmits carries to the higher order slice, and receives carries from a lower order slice. This topology creates both I/O pin difficulty and wire routing problems [Ref. 13]. Because a parallel array is highly modular, it was selected over the Wallace tree for implementation in the GSC.

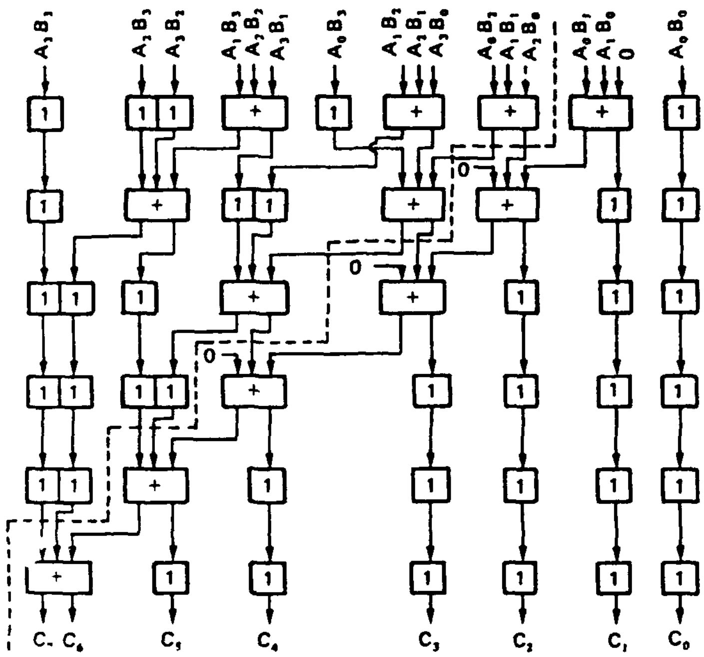


Figure 11 A Wallace Tree [From Ref. 14]

IV. PIPELINING

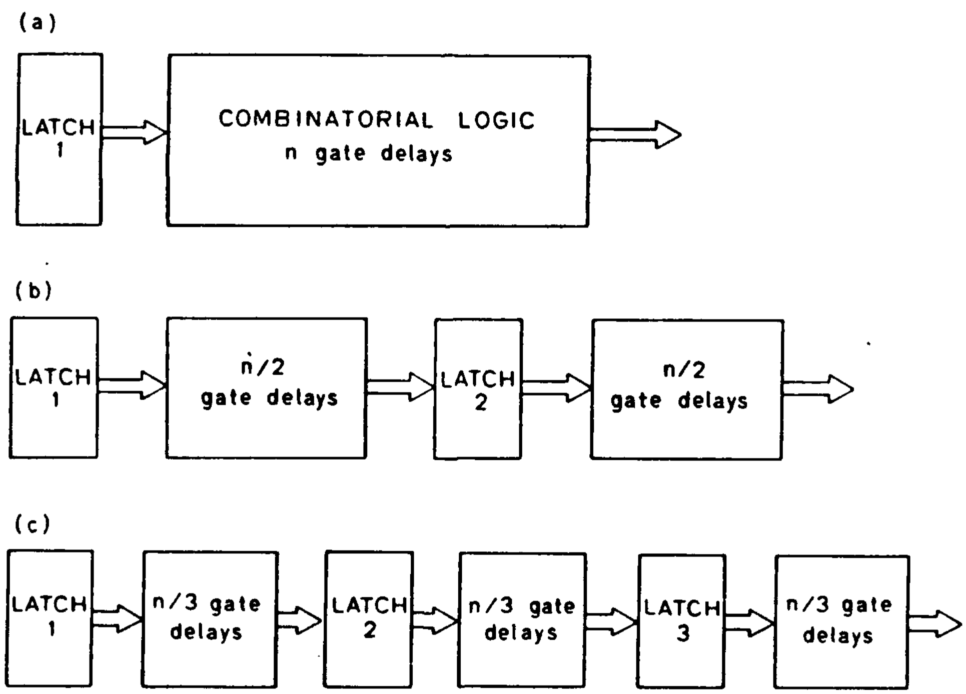
A. INTRODUCTION

The purpose of this chapter is to introduce the reader to the concept and theory of pipelining. As indicated in the title of this thesis, CMOS technology was utilized in the implementation of the parallel multiplier arrays designed in this thesis. It was previously noted that CMOS technology can substantially reduce the power consumption of a device, but results in a much slower device speed. Furthermore, it was noted that a parallel multiplier array operates at a slower speed than a multiplier tree [Ref. 13]. By incorporating pipelining into the design, however, the throughput of a parallel multiplier array may be substantially improved.

B. BASICS OF PIPELINING

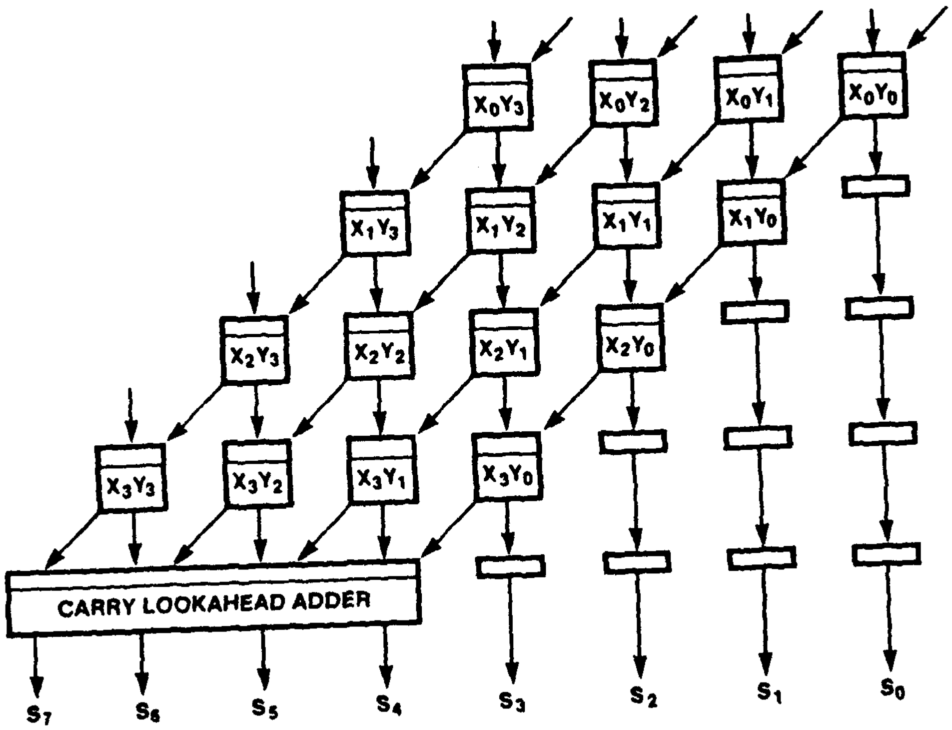
1. Bandwidth and Latency

When one reads the literature on pipelining one will observe that the term bandwidth is often associated with pipelining. Bandwidth is defined as the number of tasks that can be performed per unit time interval [Ref. 13]. For a system that operates on only one task at a time, latency is the inverse of bandwidth, and for a given latency the bandwidth can be increased by pipelining, which allows for the simultaneous execution of many tasks [Ref. 13]. Figure 12 illustrates the pipelining concept by showing that a system with latency of n gate delays can be operate at bandwidth of $1/n$, $2/n$, $3/n$, etc. Figure 13 illustrates a pipelined carry-save multiplier array; note the placement of the delay gates. This increase in bandwidth may be accomplished by dividing the combinational logic into separate stages which are in turn separated by latches [Ref. 13]. The goal of designing a multiplier using pipelining is fast operation. If some function can be executed in X ns, and the design can be separated into N stages, then a pipeline designed to perform the same function repeatedly can perform that function in times down to X/N ns [Ref. 14]. An important question one might ask regarding pipelining is what is the maximum rate at which a particular pipeline can operate. This is discussed in the following section.



Increasing bandwidth by pipelining.
a. nonpipelined system bandwidth $= 1/n$.
b. 2-stage pipelined system bandwidth $= 2/n$.
c. 3-stage pipelined system bandwidth $= 3/n$.

Figure 12 Increasing Bandwidth by Pipelining [From Ref. 13]



Pipelined carry-save multiplication array. The square boxes are carry-save adders with three latches. Each square box has three inputs: a sum and a carry from previous carry-save adders, and the third is the partial product $X_i \cdot Y_j$. The ten unmarked rectangles on the right are 1-bit latches to keep correct timing.

Figure 13 Pipelined Carry-Save Multiplier Array [From Ref. 13]

2. Analysis of a Pipelined Stage

The following definitions are commonly used in the analysis of pipelined stages:

t_x = propagation time through combinational logic (f) for this stage of the pipeline (see Figure 14 (a) and (b)).

t_r = minimum propagation time through the combinational logic (f) for this stage of the pipelining.

t_s = flip-flop setup time; the amount of time data has to be valid prior to the clocking edge.

t_h = amount of time data must be valid after clocking edge (hold time).

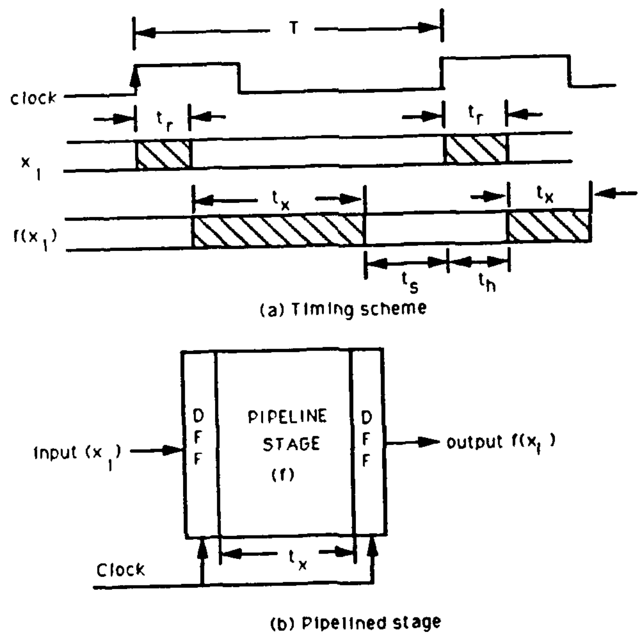


Figure 14 A Pipeline Stage

The above definitions can be used to determine the timing restrictions for a pipelined circuit. For an edge-triggered D Flip-flop;

$$\max(t_r + t_x) + t_s \leq T$$
$$\min(t_r + t_x) > t_h$$

V. DESIGN PROCESS OF A PIPELINED MULTIPLIER

A. DESIGN CONSIDERATIONS

This chapter will describe the design process for the parallel multiplier arrays implemented in this thesis. The previous sections were provided to establish a background for the design process. To gain more insight into the discussions which follow, it is highly recommended that the reader work through the tutorial section of [Ref. 8], although this is not an absolute requirement. The GSC system manuals include a tutorial section. However, this author believes it was written with the presumption that the reader had attended a one-week course of instruction taught by the Silicon Compiler System Corporation of San Jose, California. Without this course of instruction the user may have some difficulty working through the tutorial sections until some proficiency has first been acquired.

As stated earlier, the parallel multiplier array of Figure 8 (incorporating the parallel multiplier cell) was selected for implementation in the GSC. This decision was based primarily on the array's modular architecture. It was also apparent that its feature of horizontal and vertical feedthrough was advantageous for implementation in VLSI because the routing of the inputs X_i and Y_i throughout the entire array would be simplified.

1. Modeling the Parallel Multiplier Cell

One of the first design considerations contemplated was how to model the basic parallel multiplier cell of Figure 8. In Figure 8, the bit-wise ANDing of the partial products occurs inside the cell's boundaries. The results of each bit-wise AND is summed with the SUM of another multiplier cell, as well as with a

CARRY IN. The author determined that this cell could be implemented in GENESIL by using a 1-bit full adder with one input being provided by the output of an AND gate (from the formation of the partial products) and the other from the SUM of another adder. Note that a 1-bit full adder also provides for a CARRY IN and CARRY OUT. Figure 15 shows the basic cell and its layout is illustrated in Figure 16.

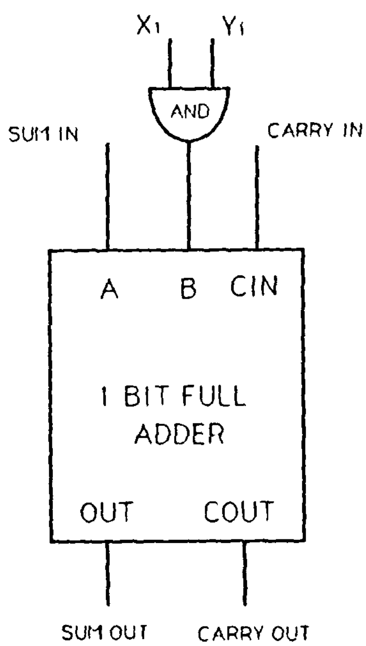


Figure 15 Parallel Multiplier Cell for Implementation in GENESIL

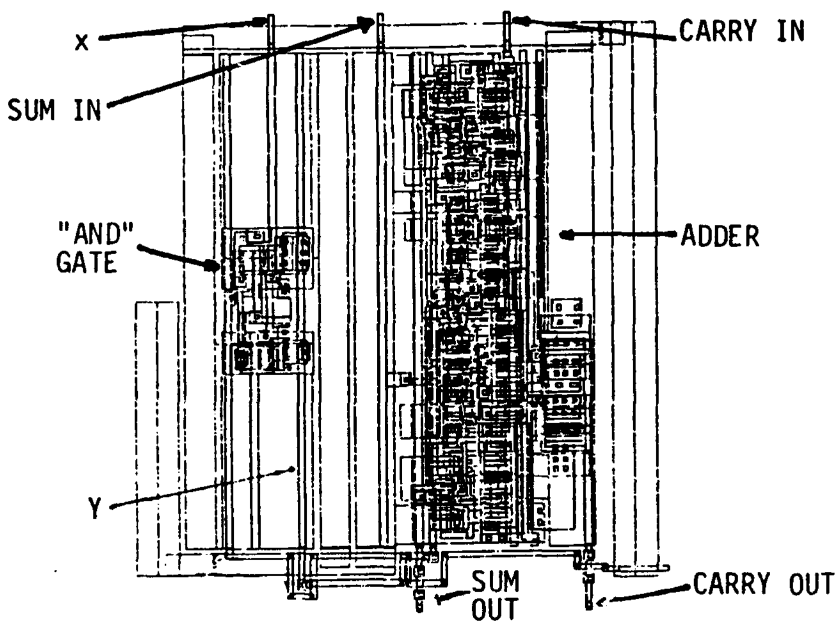


Figure 16 GENESIL Layout of a Parallel Multiplier Cell

(101.6 mils²)

2. Selecting a Fabline

The next design consideration was to select a "fabline", that is, a particular set of design rules used by a foundry to manufacturer a Chip. Because Stuart [Ref. 15] did a full custom parallel multiplier array design using 1.5 CMOS, the same micron technology was selected for this study to enable a comparison of results. Figure 17 shows the fablines available for selection.

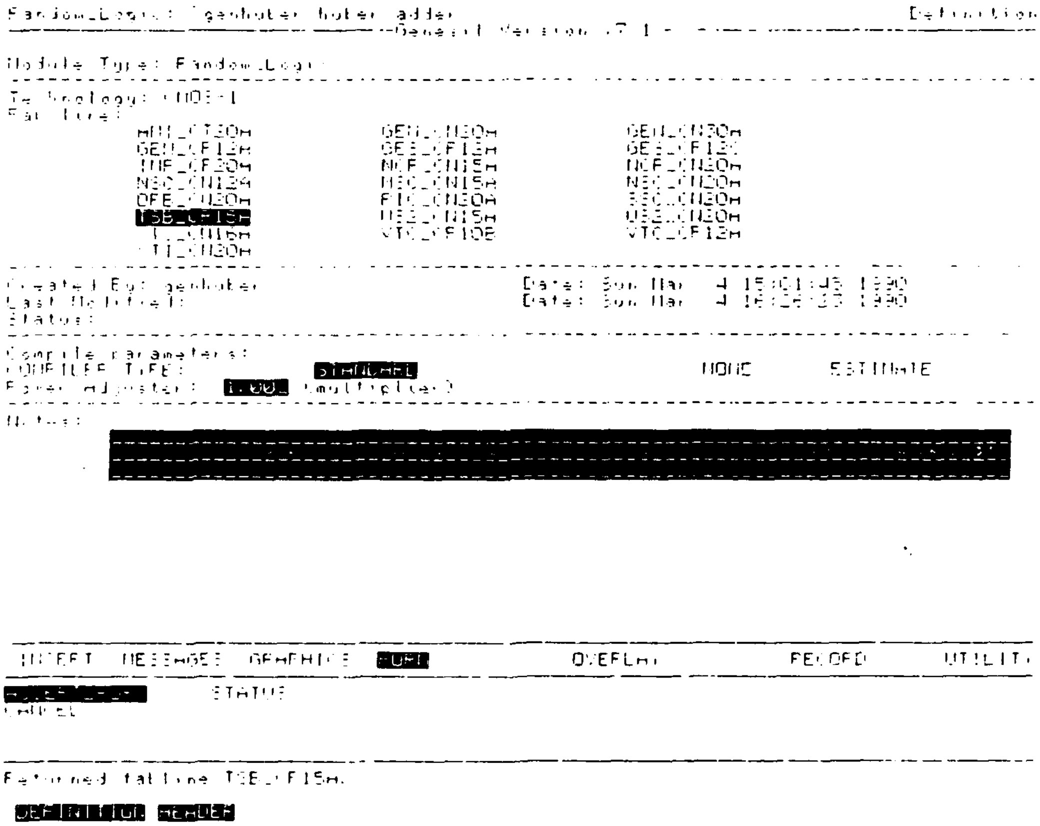


Figure 17 Selection of a Fabline

Note that fablines which include the number 15 are 1.5 µm technology. To assist in the selection of a particular 1.5 CMOS fabline speed was used as the criterion. To determine which fabline was the fastest, a timing analysis was performed on four adders each incorporating a different 1.5 µm fabline. Figure 18 illustrates a linear view of a GENESIL 1-bit full adder (note the labeling of the signal lines), and Figure 19 illustrates the layout of a 1-bit GENESIL full adder.

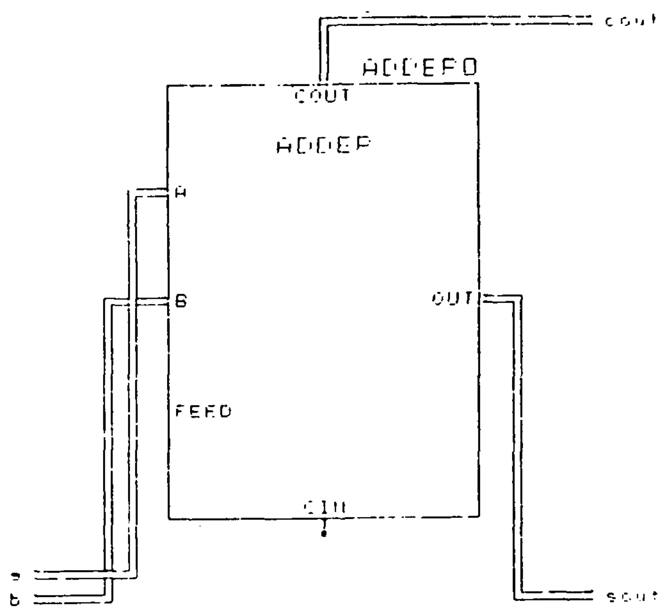


Figure 18 Linear View of a GENESIL 1-Bit Full Adder

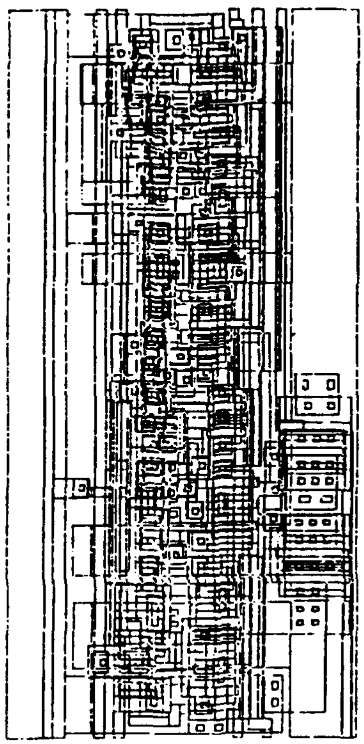


Figure 19 GENESIL Layout of a 1-Bit Full Adder

The results of the timing analysis are listed in Table 1. The NSC_CN15A fabline was selected because it had the smallest maximum output delay for both the CARRY OUT (cout[0]) and the SUM OUT (sout[0]).

TABLE 1

OUTPUT DELAYS FOR A GENESIL 1-BIT FULL ADDER

	cout[0] Ph1 (r) Delay(ns)		sout[0] Ph1 (r) Delay(ns)				
Fabline	Min	Max	Min	Max	height (mils)	width (mils)	area (mils^2)
TSB_CP15A	2.8	7.2	2.8	7.2	8.91	4.28	38.08
NCR_CN15A	3.5	8.4	3.5	8.4	8.91	4.28	38.08
US2_CN15A	3.5	8.1	6.3	7.5	10.09	4.85	48.91
NSC_CN15A	2.1	5.1	3.9	4.9	8.91	4.28	38.08

Note: 1 mil = 0.001 inches

In addition to the 1-bit full adder, a GENESIL D flip-flop was also tested to determine if there was a difference in the output delay for each 1.5 μm fabline. The results are listed in TABLE 2. As expected, in view of the results in TABLE 1, the NSC_CN15A fabline produced a shorter output delay than the other fablines. Figure 20 illustrates a linear view of a GENESIL D flip-flop and Figure 21 illustrates the GENESIL layout of a D flip-flop.

TABLE 2

OUTPUT DELAY FOR A GENESIL D FLIP-FLOP

	Ph1 (r) Delay(ns)				
Fabline	Min	Max	height (mils)	width (mils)	area (mils^2)
TSB_CP15A	4.5	5.0	3.27	8.46	27.63
NCR_CN15A	6.0	6.2	2.88	7.46	21.51
US2_CN15A	4.8	5.8	2.88	7.46	21.51
NSC_CN15A	3.8	4.0	2.88	7.46	21.51

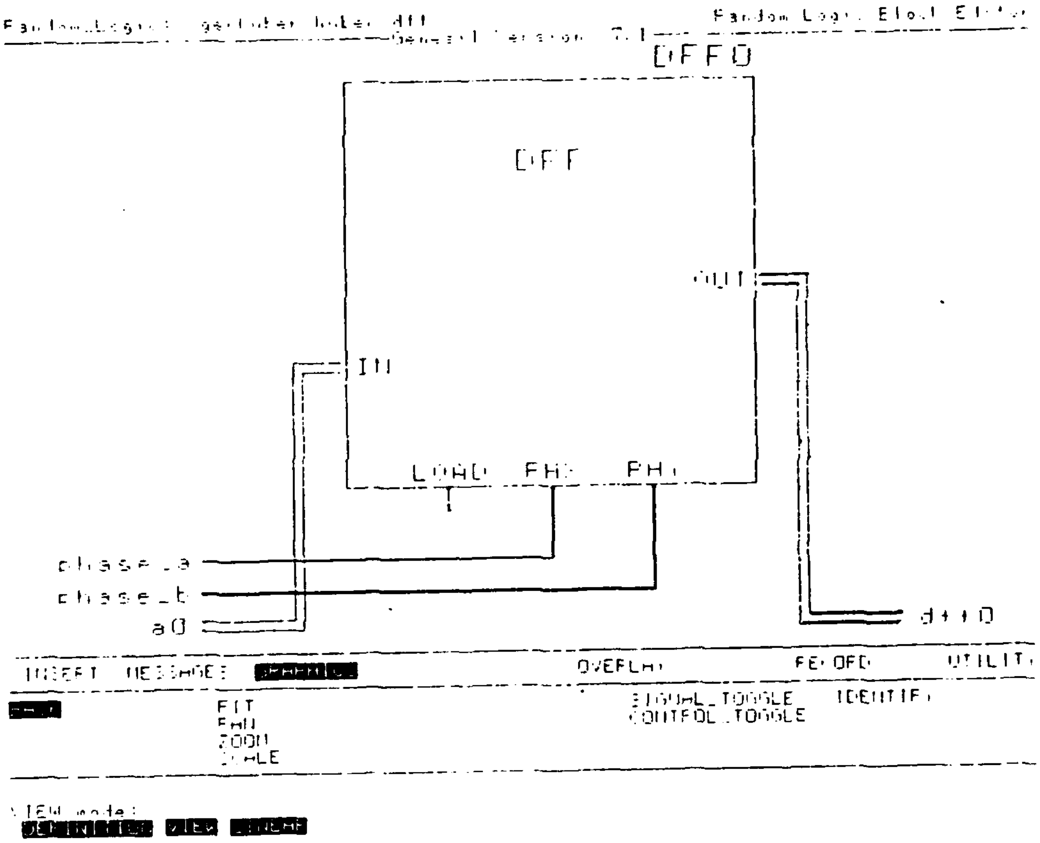


Figure 20 Linear View of a GENESIL D Flip-Flop

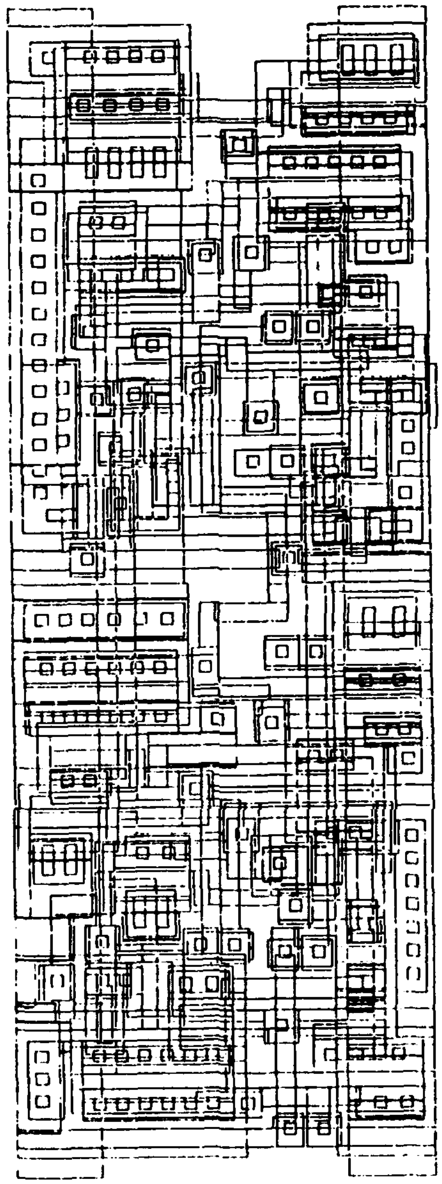


Figure 21 GENESIL Layout of a D Flip-Flop

The following section will begin describing the design process and the integration of the parallel multiplier cells into functional multiplier arrays.

B. DESIGN OF A 4-BIT PIPELINED MULTIPLIER ARRAY

1. Signal Naming Scheme

The author made a decision early in the implementation phase to first demonstrate the feasibility and functionality of the parallel multiplier array by constructing a 4-bit unsigned multiplier. Once the basic design was validated, a pipelined version and larger arrays were then constructed.

Using a CAD, a 4-bit version of Figure 9 was drafted and is shown in Figure 22. However, before the drawing could be made it was necessary to devise a signal naming scheme. A requirement was set that this scheme must impart some information on the origin of a signal, to assist in trouble shooting the circuit, as well as be applicable to all of the parallel multipliers implemented in this thesis.

Therefore, the scheme was based on a labeling convention similar to that of a full adder. For example, the signals SUM OUT and CARRY OUT were labeled as product out "po" and carry out "co", respectively. These labels were further modified to "po*kj*" and "co*kj*", where k indicates the level number and j indicates the adder position in a particular level. Here, k ranges from 0 to n , where n is the number of bits the multiplier is capable of operating on. The j indicates the position of the adder from the right-hand side of the level in which it is located and it ranges from 0 to $n - 1$. For example, "po23" indicates the signal "product out" from level 2 adder 3. Additionally, all AND gates were labeled according to the partial products they form. For example, X_2Y_0 indicates the ANDing of the partial products X_2 and Y_0. Furthermore, each row of adders were labeled as "level_k" and each adder was labeled as "ADDkj", where k and j correspond to

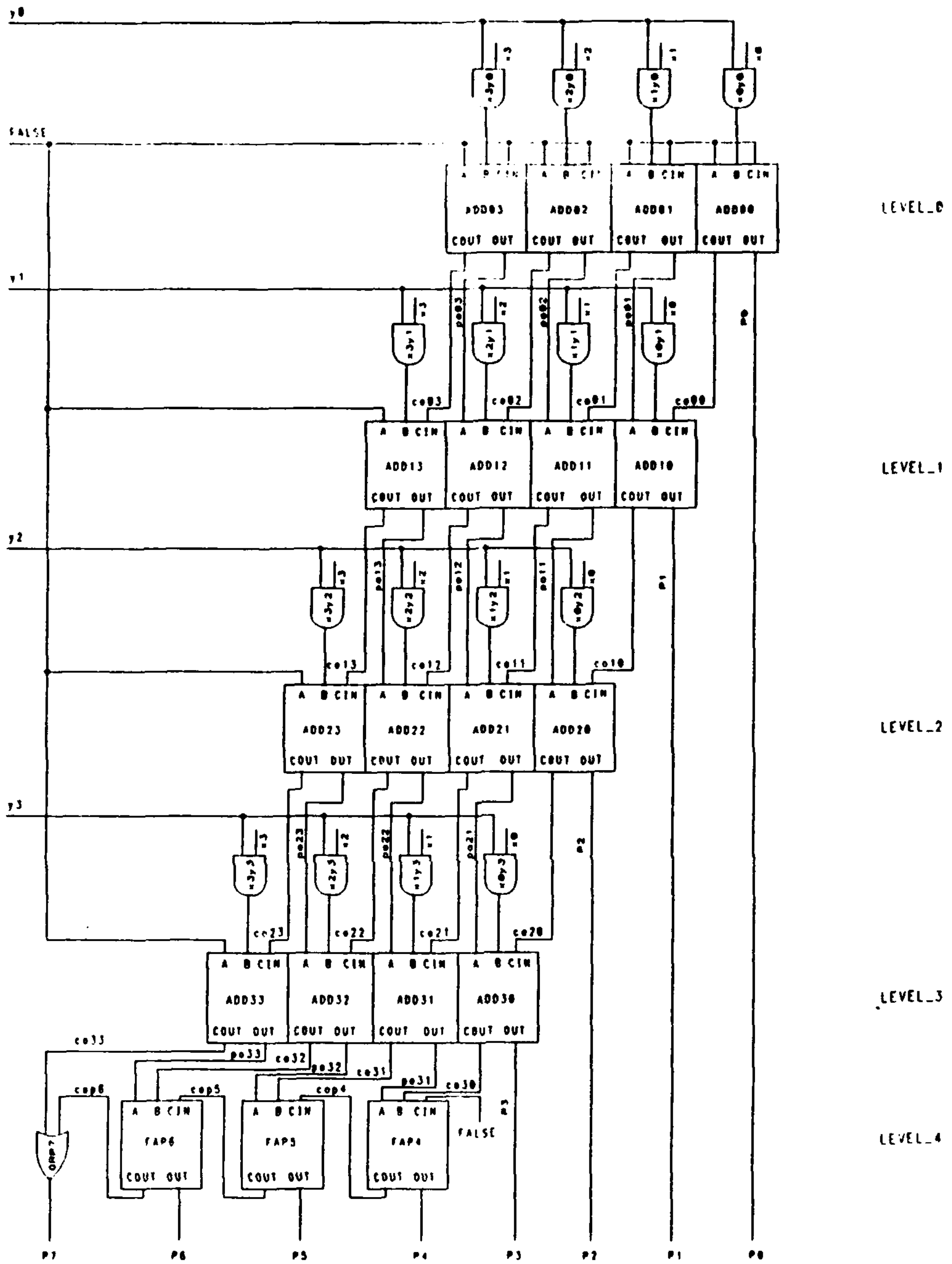


Figure 22 CAD Layout of a 4-Bit Parallel Multiplier Array

the level number and the adder's position, respectively. Finally, the last row of adders was labeled as "FAP*x*" where *x* indicates a particular final product. For example, "FAP4" indicates the final adder whose output is product 4.

2. 4-Bit Multiplier Array

From the very start of the construction phase for the 4-bit multiplier array, there were questions regarding what method(s) and what Blocks or Modules should be employed to build the arrays. The first approach at constructing the array was to create a random logic Block (labeled multi_4bit). After selecting the fabline NSC_CN15 for this Block, 19 full adders, 16 AND gates, and one OR gate were attached to it through the use of the options SPECIFICATION and NEW. These components were then connected as in Figure 22 by indicating the appropriate signal names in the SPECIFICATION form. The SIGNALS function was then used to designate whether a particular signal was an "input, output or bi-level." This first attempt resulted in a long "stick-like" structure (see Figure 23) which would not be suitable for a Chip layout simply due to its inefficient use of space. If larger multipliers were constructed using this method one would produce long arrays whose length would be proportional to the number of bits to be multiplied. Therefore, other methods were sought to reduce the length of the array.

One method considered was to simply divide the array into rows of adders (similar to Figure 10) according to their level by putting each row of adders in random logic Block. Each random logic Block would then be attached to a general random logic Module (labeled 4bmm; for 4-bit multiplier module) and the rows of adders would be interconnected again as in Figure 22. When implemented, this method proved successful in reducing the previous "stick-like" structure to a more compact modular arrangement. Figure 24 is the GENESIL layout of this new modular arrangement.

Figure 23 GENESIL Layout of multi_4bit

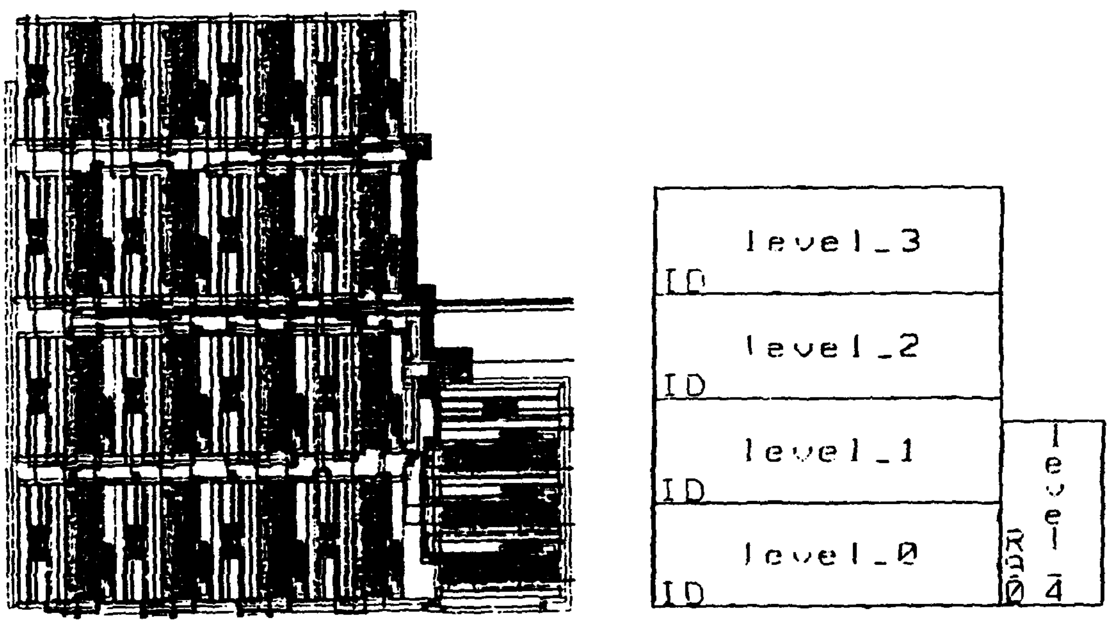


Figure 24 GENESIL Layout of 4bmm (1,958.3 mils^2)

The construction of the rows of adders (levels) in the modular arrangement was accomplished through the employment of a generic "level_k". As stated previously, a random logic Block was defined and four adders and four AND gates were attached to it. The Block was then label as level_k. Through the use of "ATTACH EXISTING", while the Module 4bmm was at the top of the hierarchy, the generic level_k was successively attached. Each time level_k was attached to the Module it was renamed according to it assigned level in Figure 22. The last row of adders was constructed by simply deleting the AND gates and 1-bit full adder from the generic level_k, and attaching an OR gate. The generic level_k is illustrated in Figure 25. Figure 26 is a GENESIL linear view of the generic level_k. A CAD drawing of the general random logic Module 4bmm illustrating its block level layout is shown in Figure 27.

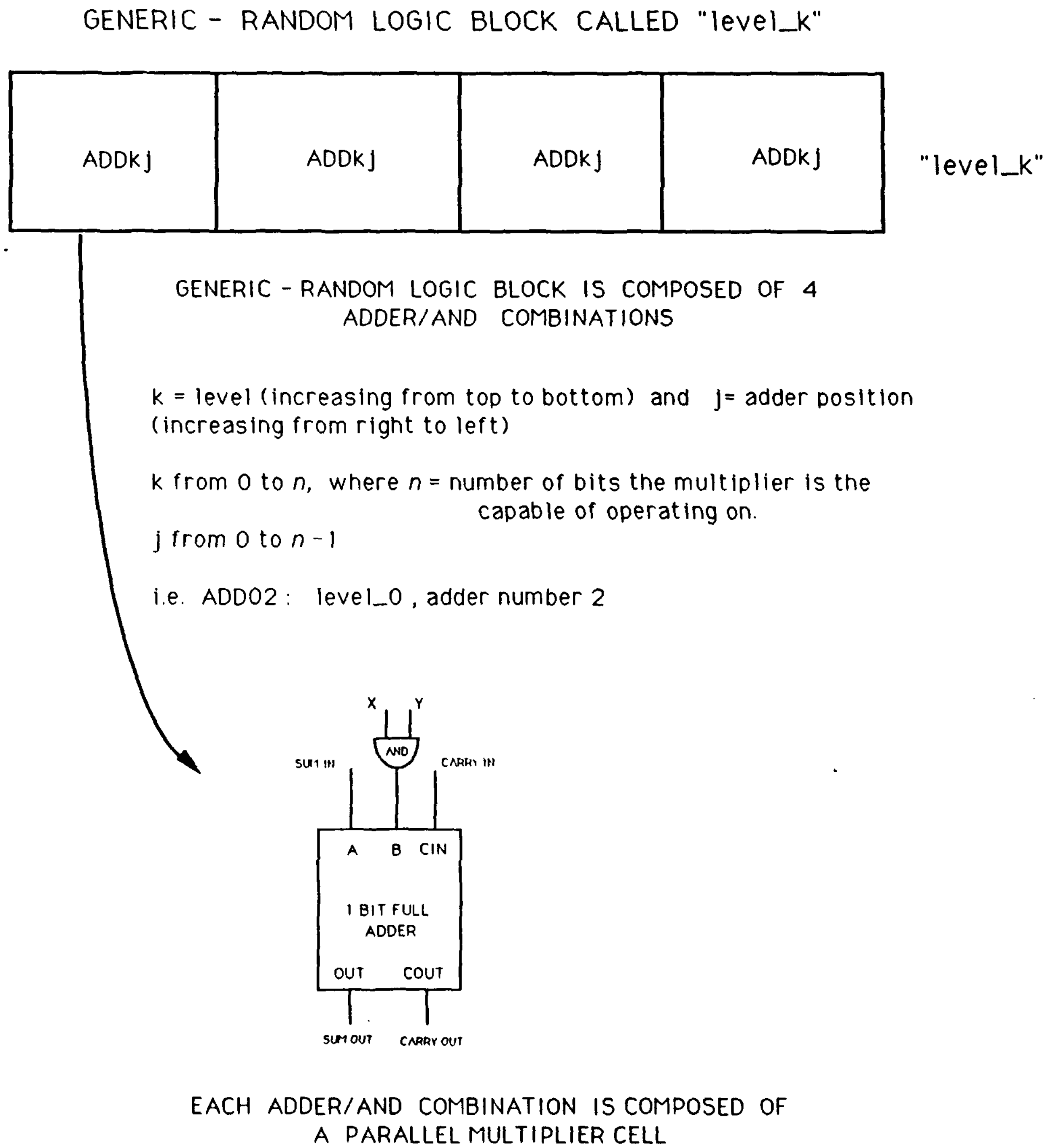


Figure 25 CAD Depiction of Generic Level_k

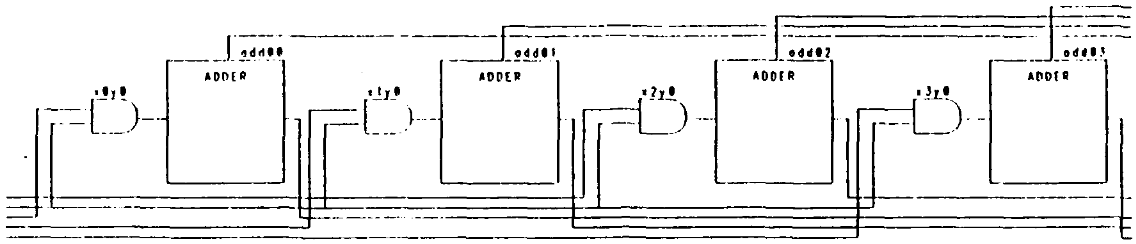


Figure 26 GENESIL Linear View of Generic Level_k

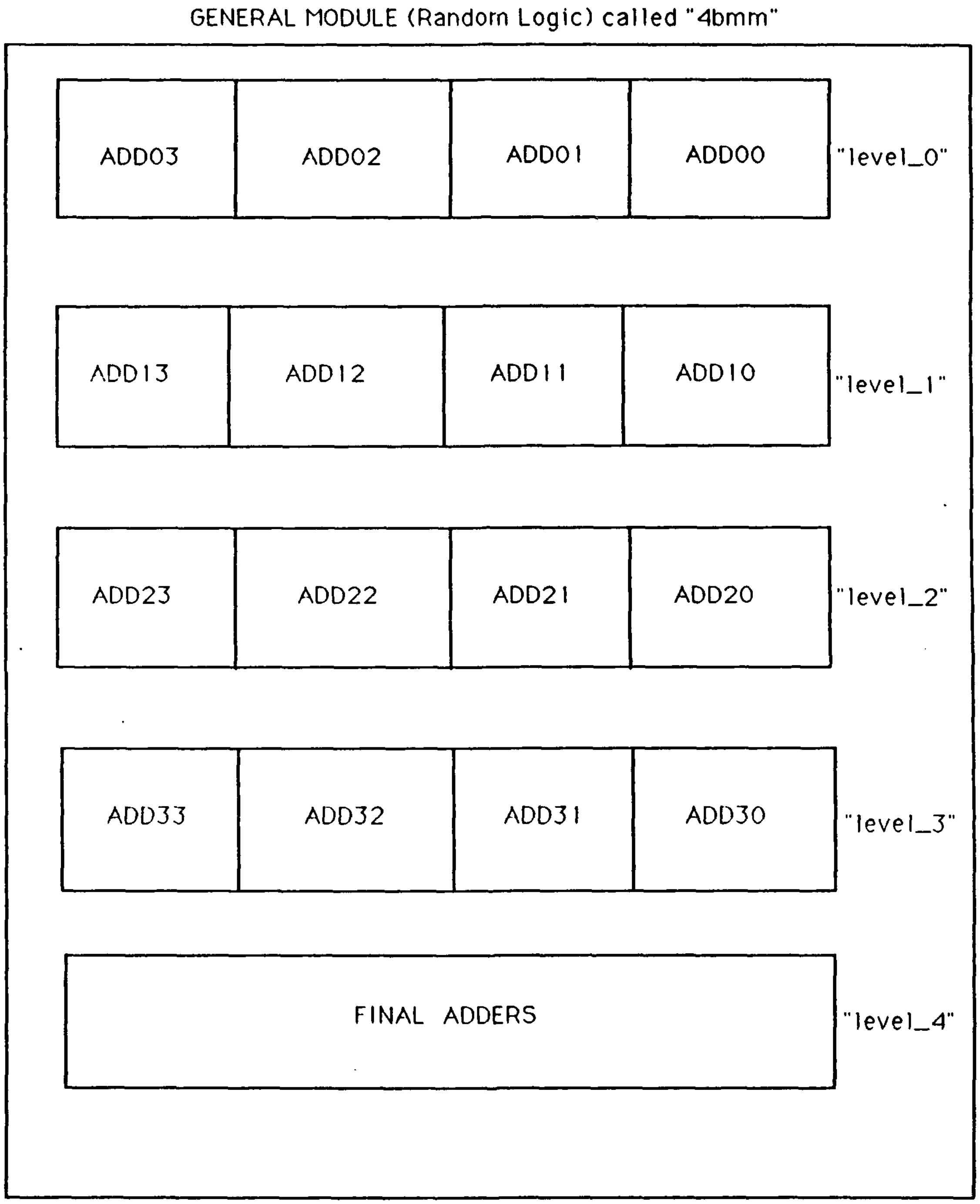


Figure 27 General Module 4bmm

A. Version 1

After a close inspection of Figure 24 (from this point on this layout will be referred to as 4bmm.1 to indicate version 1 of 4bmm) the author decided that the modular arrangement of Figure 27 was probably the best one to use when implementing parallel multiplier arrays in GENESIL. This decision was based primarily on the modular arrangement of the parallel multiplier cells, as well as the overall symmetry of the layout.

Before attempting to improve on the initial layout of Figure 24, the functionality of the multiplier array was verified. This was a simple task and was accomplished as described on page 102 of Reference 7. Several different binary numbers were multiplied and their resulting products were verified using a hand-held HP-28S calculator. The following is an example of how multiplication was performed by GSC. The assignment of binary values to the inputs of 4bmm.1, x[3:0] and y[3:0], and the product of multiplication is illustrated in Figures 28 and 29, respectively.

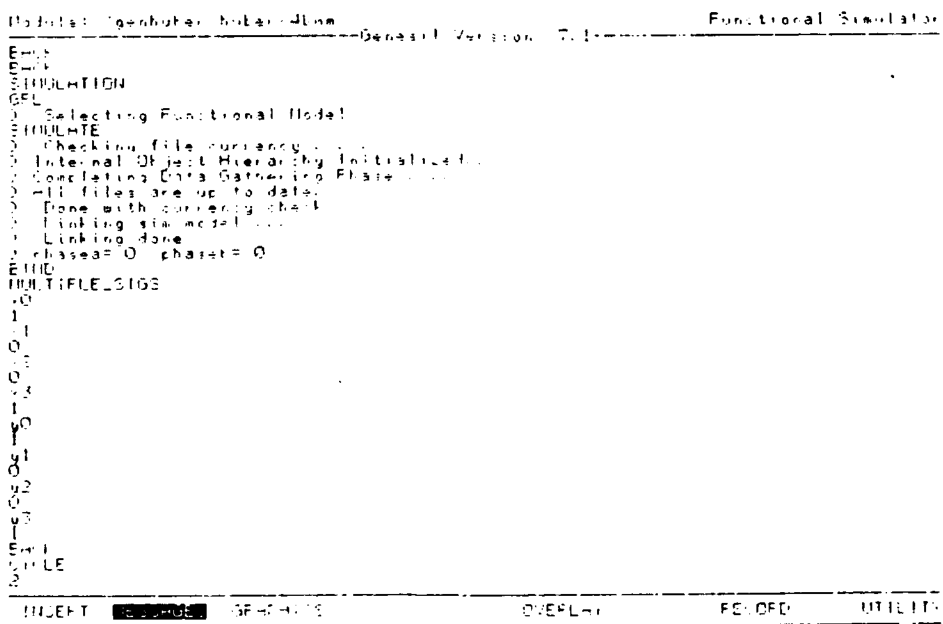


Figure 28 Assignment of Binary Values to Inputs of 4bmm.1

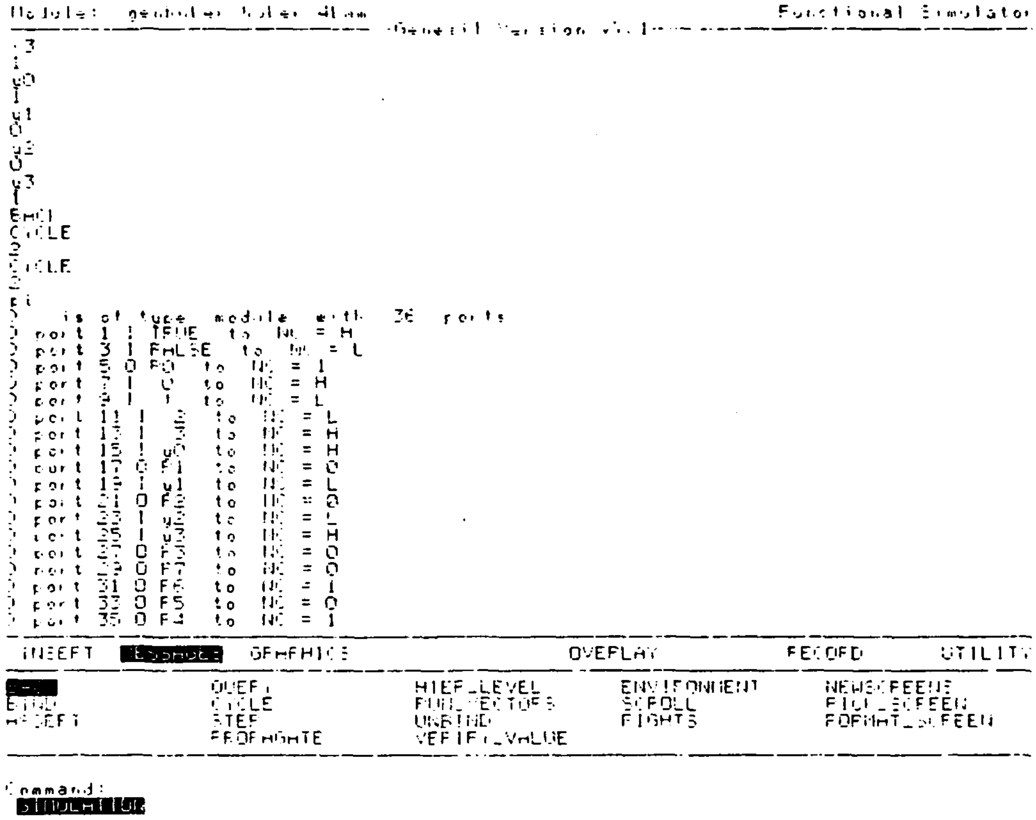


Figure 29 Product of Multiplying 1001x1001 Using 4bmm.1

Following the verification of the functionality of 4bmm.1, a timing analysis was performed to determine the output delays for each product P[7:0]. This was accomplished by selecting TIMING from the Executive menu and executing OUTPUT_DELAY. The results are listed in Figure 30 and indicate 4bmm.1 can theoretically be operated at approximately 29 MHz (1/34.7 ns). This calculation is based on the output delay of P7 since it is the limiting product; it has the largest maximum delay of the other products.

Once 4bmm.1 was verified to be operating correctly, attempts were made to improve the speed and reduce the size of the array, by experimenting with changing the order and the location of the adder levels and by replacing "FAP4-6" with a GENESIL library 3-bit adder.

B. Version 2

Version two of the array was created by replacing the final adders of level_4 (FAP4-6) with a GENESIL library 3-bit adder (see Figure 31). As in version one, a functional verification was conducted first before performing a timing analysis. The results of the timing analysis are listed in Figure 32 and the layout of 4bmm.2 is shown in Figure 33. One can see from the results in Figure 32 that the use of the GENESIL library 3-bit adder in level_4 resulted in a slight reduction in the output delay for P7. The operating speed was calculated to be approximately 30 MHz, and there was no significant change in size. However, comparing the layout of level_4 of version 1 and 2 shows that the GENESIL 3-bit adder of version 2 is of higher density than the 3 individual 1-bit adders of version 1.

```
[illegible]                                                   Timing Analyzer
---------------------------- [illegible] --------------------------------
OUTPUT DELAY MODE
--------------------------------------------------------------------------
[illegible]                                      [illegible]
  Junction Temperature:75 degree C               Voltage:5.00
  Phase 1:   ---                                 Phase 2:   ---
 Included setup files: default setup file
--------------------------------------------------------------------------
                              OUTPUT DELAY (ns)
Output                PH1(r) Delay        PH2(r) Delay        Loading(pf)
                      Min      Max        Min      Max
```

Output	PH1(r) Delay Min	PH1(r) Delay Max	PH2(r) Delay Min	PH2(r) Delay Max	Loading(pf)	
P0	5.7	5.5	[illegible]	5.5	0.10	FHTH
P1	[illegible]	11.[illegible]	[illegible]	11.[illegible]	0.10	FHTH
P2	[illegible]	[illegible]	[illegible]	[illegible]	0.10	FHTH
P3	[illegible]	[illegible]	[illegible]	[illegible]	0.10	FHTH
P4	[illegible]	[illegible]	[illegible]	[illegible]	0.10	FHTH
P5	[illegible]	[illegible]	[illegible]	[illegible]	0.10	FHTH
P6	[illegible]	[illegible]	[illegible]	[illegible]	0.10	FHTH
P7	[illegible]	[illegible]	[illegible]	[illegible]	0.10	FHTH

Figure 30 Timing Analysis of 4bmm.1

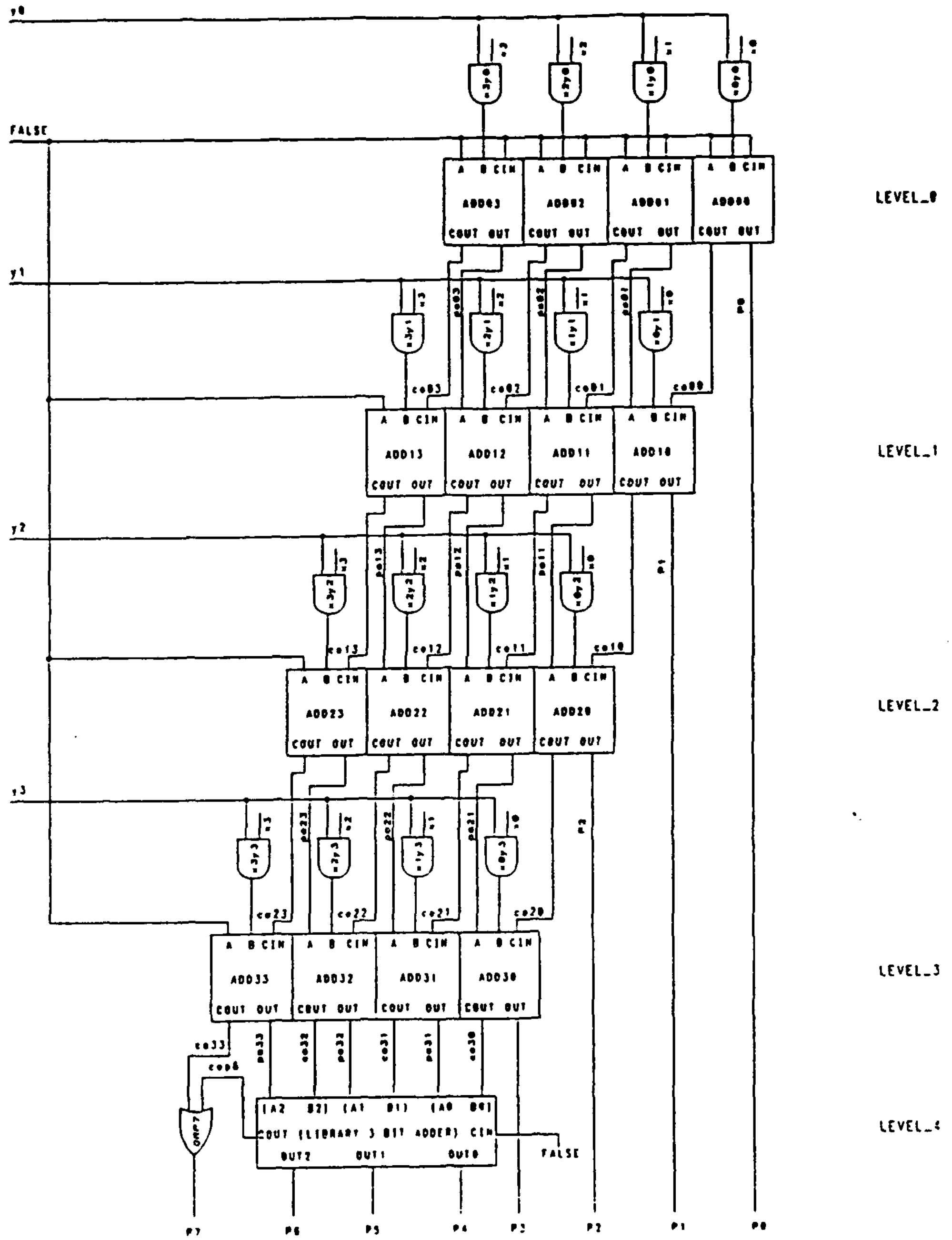


Figure 31 CAD Layout of 4bmm.2

Module: [illegible] 4bmm.2Es — Genesil Version v7.1 — Timing Analyzer

OUTPUT DELAY MODE

Fab line: NSC_CN15m — Corner: TYPICAL
Junction Temperature: 75 degree C — Voltage: 5.00V
Phase 1: --- — Phase 2: ---
Included setup files: default setup file

OUTPUT DELAYS (ns)

Output	Ph1(r) Delay Min	Ph1(r) Delay Max	Ph2(r) Delay Min	Ph2(r) Delay Max	Loading(pf)	
F0	5.7	6.5	5.7	6.5	0.10	FMTH
F1	5.5	11.5	5.5	11.8	0.10	FMTH
F2	5.5	17.[illegible]	5.5	17.[illegible]	0.10	FMTH
F3	5.5	[illegible]	5.5	[illegible]	0.10	FMTH
F4	[illegible]	[illegible]	[illegible]	[illegible]	0.10	FMTH
F5	[illegible]	[illegible]	[illegible]	[illegible]	0.10	FMTH
F6	[illegible]	[illegible]	[illegible]	[illegible]	0.10	FMTH
F7	[illegible]	[illegible]	[illegible]	[illegible]	0.10	FMTH

Figure 32 Timing Analysis of 4bmm.2

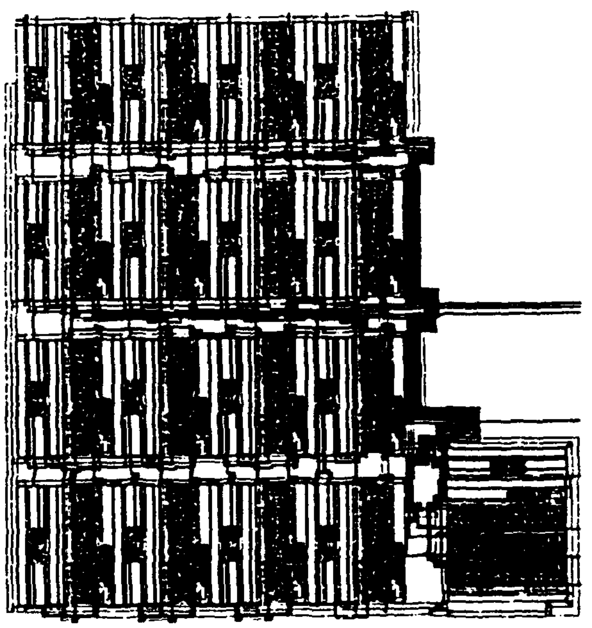


Figure 33 GENESIL Layout of 4bmm.2 (1,964.02 mils²)

C. Version 3

Version 3 (4bmm.3) was the first attempt at reordering the adder levels to determine what effect this would have on the size and speed of the array. When developing versions 1 and 2, the ordering of the levels was determined by the AUTO_PLACEMENT option from the PLACEMENT menu which is a submenu of FLOORPLANNING. Although the specifications of the array were entered into the GSC as in Figure 22, this did not necessarily guarantee that the levels would be oriented in the same manner. When performing FLOORPLANNING the user can elect to use either AUTO_PLACEMENT or manual PLACEMENT to arrange the relative positions of the levels. For versions 1 and 2 AUTO_PLACEMENT was selected. It uses an algorithm built into the GSC to determine the best placement of the individual levels. Figure 34 illustrates the AUTO_PLACEMENT of the adder levels as determined by the GSC. Note that the order is arranged according to the specifications of Figure 22, with the exception that the final adders (level_4) are located to the right of level_0.

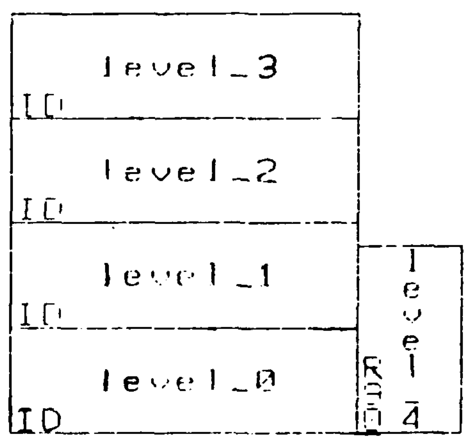


Figure 34 AUTO_PLACEMENT of Adder Levels (V1&2)

In version 3 (4bmm.3) the order was rearranged from top to bottom, using manual PLACEMENT, according to the "logic flow". This reordering is illustrated in Figure 35. Note that the final 3-bit adder (level_4) is now located below level_3. A GENESIL layout of this arrangement is shown in Figure 36.

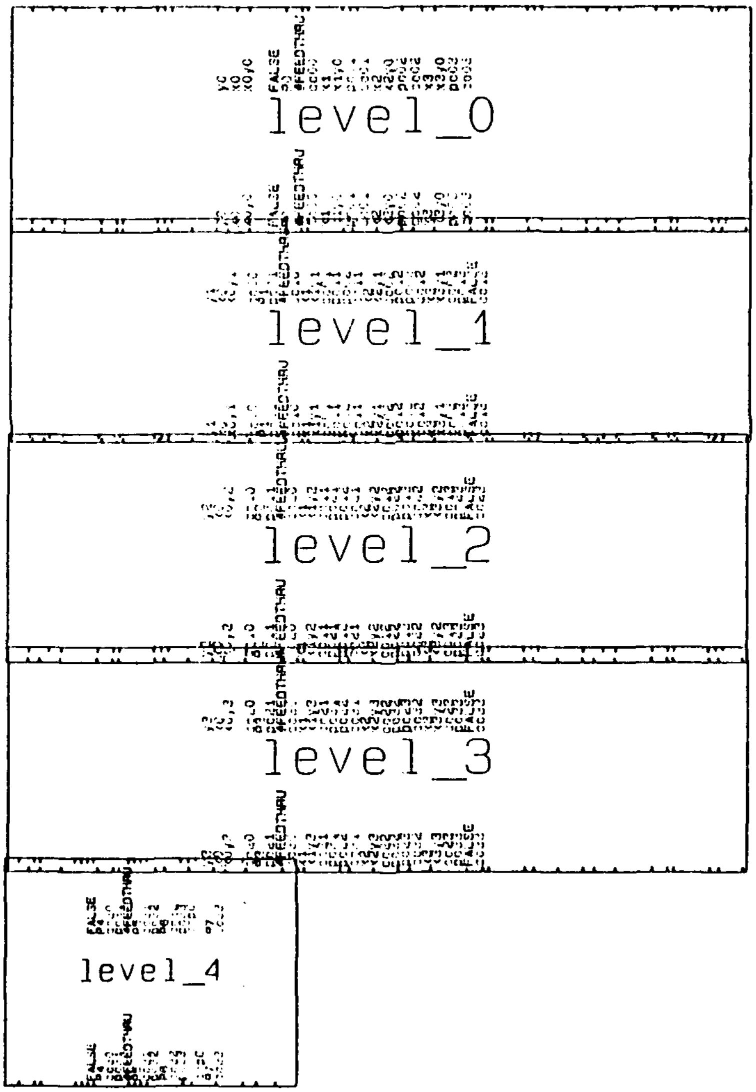


Figure 35 Reordering of Adder Levels According to Logic Flow

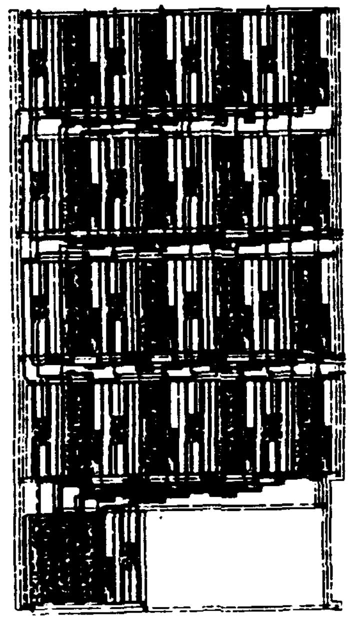


Figure 36 GENESIL Layout of 4bmm.3 (1,845.63 mils^2)

From the results of a timing analysis performed on 4bmm.3 it was determined that the reordering had no significant effect on the output delay of P7. The output delay for P7 of 4bmm.2 was 32.5 ns and for 4bmm.3 it was 32.4 ns. However, there was a 6% reduction in the overall size of the array. The 4bmm.2 design had total area of 1964.02 mils^2 while that of 4bmm.3 was calculated to be 1845.63 mils^2. Close inspection of Figure 36 reveals that there is almost an equal distribution of metal above the final adders of level_4. One can see metal stretching from the lower right side of level_3 across to the adders of level_4. Level_4 was centered directly below level_3 to see if the metal routing could be more equally distributed and perhaps further reduce the total area. This was accomplished in version 4 below.

D. Version 4

As stated above, version four (4bmm.4) was simply a centering of level_4 directly below level_3. The layout of 4bmm.4 is shown in Figure 37. Again, there was no further reduction in the output delay of P7, however, there was a very slight reduction in the size of the array. The total area of 4bmm.4 was calculated to 1835.9 mils^2 which is a 1% reduction in the total area of 4bmm.3. Also, note that the metal routing between levels 3 and 4 has been thinned out.

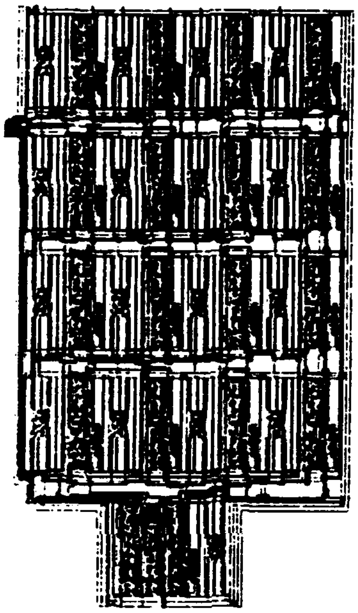


Figure 37 GENESIL Layout of 4bmm.4 (1,835.9 mils^2)

3. 4-Bit Multiplier Array with Registered Inputs/Outputs

A. Version 1

When multipliers are implemented in actual circuits they are often constructed with registered inputs and outputs. This is essential for pipelined multipliers. Therefore, a bank of 8 D flip-flops was added to the inputs, x[3:0] and y[3:0], and to the products P[7:0] as illustrated in Figure 38 (labeled 4bmm1.RIRO). Here, AUTO_PLACEMENT was used to see what the GSC system would determine to be the best placement of the adder levels and the two banks of D flip-flops. The resulting floorplan is shown in Figure 39. Note

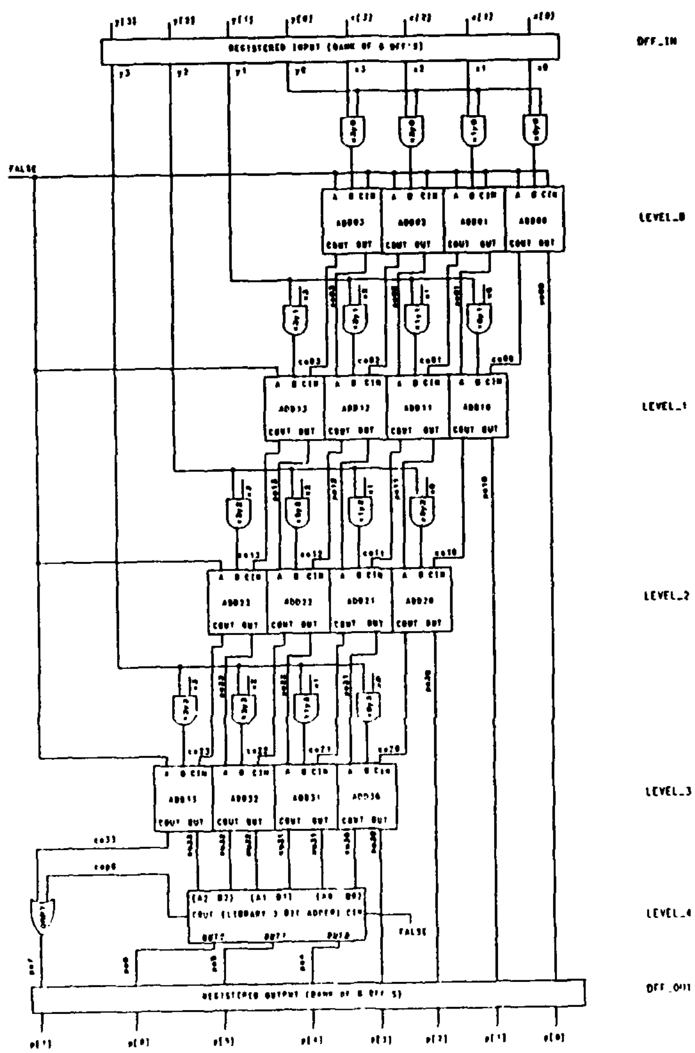


Figure 38 CAD Drawing of 4bmm1.RIRO

how the AUTO_PLACEMENT algorithm placed the input registers next to the level_3 adders. One can see similarities here between the floorplans of 4bmm.1 and 4bmm.2 of Figure 34. It appears the AUTO_PLACEMENT algorithm favors the placement of level_4 next to level_0. Figure 40 is a GENESIL layout of 4bmm1.RIRO.

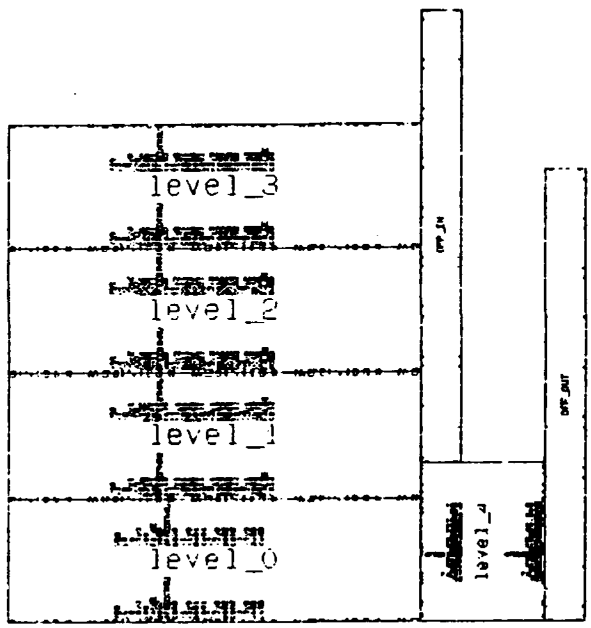


Figure 39 AUTO_PLACEMENT of 4bmm1.RIRO

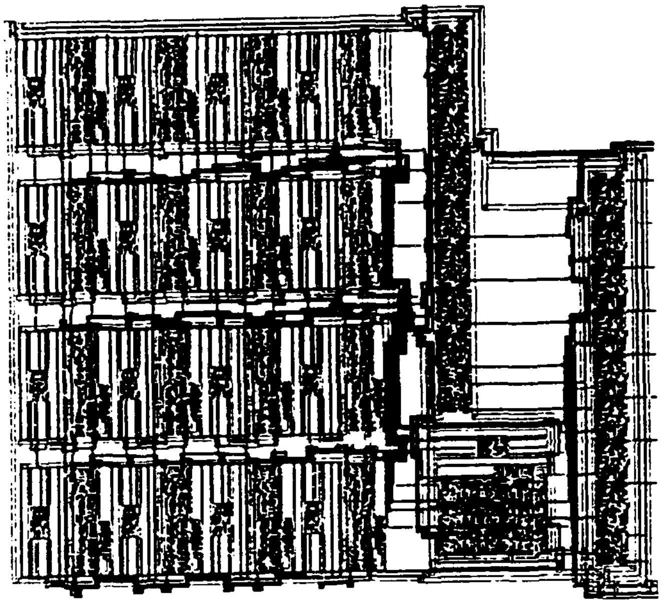


Figure 40 GENESIL Layout of 4bmml.RIRO (2,551.69 mils^2)

B. Version 2

Version 2 (4bmm2.RIRO) is 4bmm.4 with registered inputs and outputs. It was implemented in the same fashion as 4bmml.RIRO, however, manual PLACEMENT was used instead of AUTO_PLACEMENT. The input and output registers were manually placed as drawn in Figure 38, and the resulting floorplan is illustrated in Figure 41. Here, one can see an overlap between adjacent levels. The was done manually to determine what effect overlap would have on the GSC. The resulting layout of 4bmm2.RIRO is shown in Figure 42. The total area of 4bmm2.RIRO was 2459.07 mils^2 while 4bmml.RIRO totaled 2551.69 mils^2. The 4bmm2.RIRO design resulted in approximately a 3.6 % reduction in area compared to 4bmml.RIRO, and had a much "cleaner" looking layout.

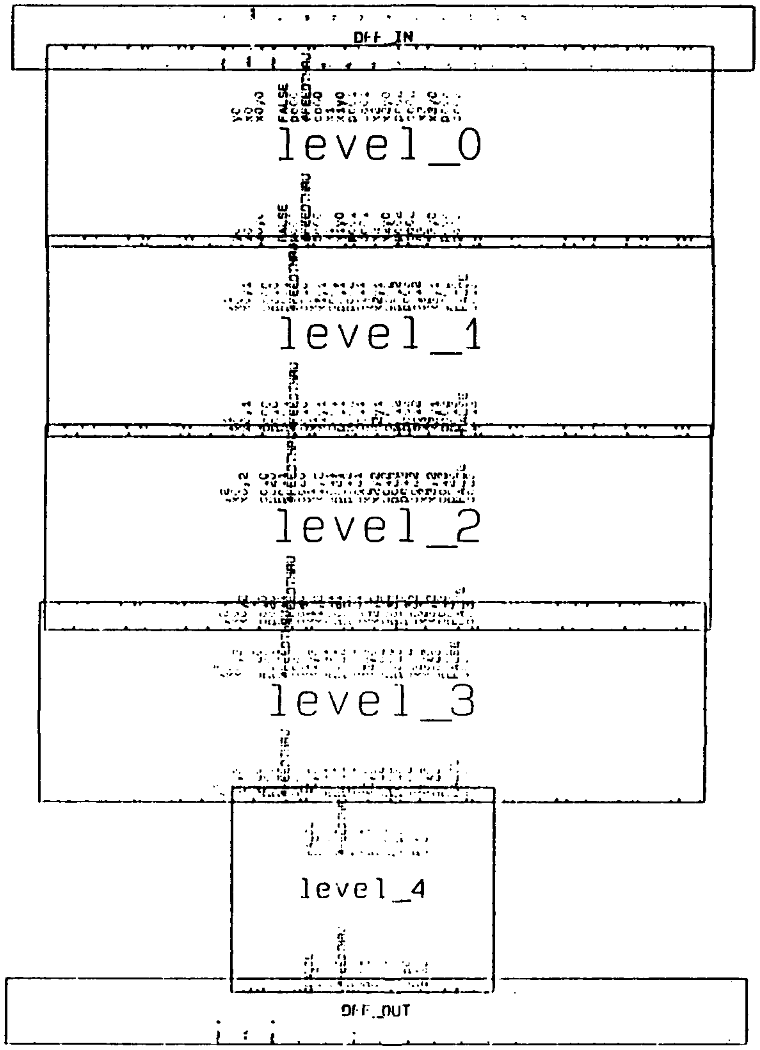


Figure 41 Floorplan for 4bmm2.RIRO

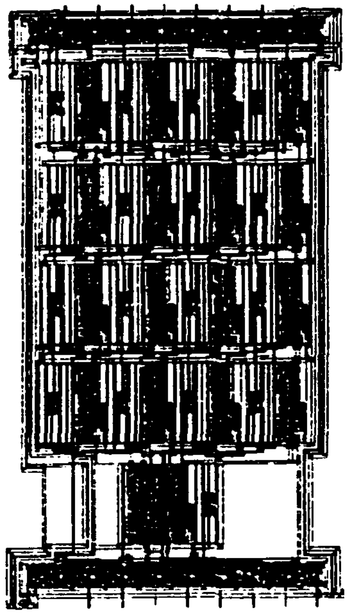


Figure 42 GENESIL Layout of 4bmm2.RIRO (2,459.07 mils^2)

4. 4-Bit Pipelined Multiplier Array

After experimenting with the 4-bit multiplier array, the author concluded that the best arrangement for the registers and adder levels was as indicated in Figure 42. As demonstrated by the timing analysis for 4bmm.2 and 4bmm.3, there was no significant reduction in the output delay of P7 when the adder levels were oriented in the order of "logic flow". However, it was demonstrated that orienting the adder levels in the order of the "logic flow" resulted in an overall reduction in array area. With this in mind, it was decided to orient the pipelined version of the 4-bit multiplier array in the same manner; that is, in the order of the "logic flow."

Before designing the 4-bit pipelined version it was necessary to determine between what levels to insert a bank of D flip-flops. From inspection of Figure 32, it was decided to insert a row of flip-flops between level_2 and level_3 (see Figure 43). This would provide for two pipelined stages without

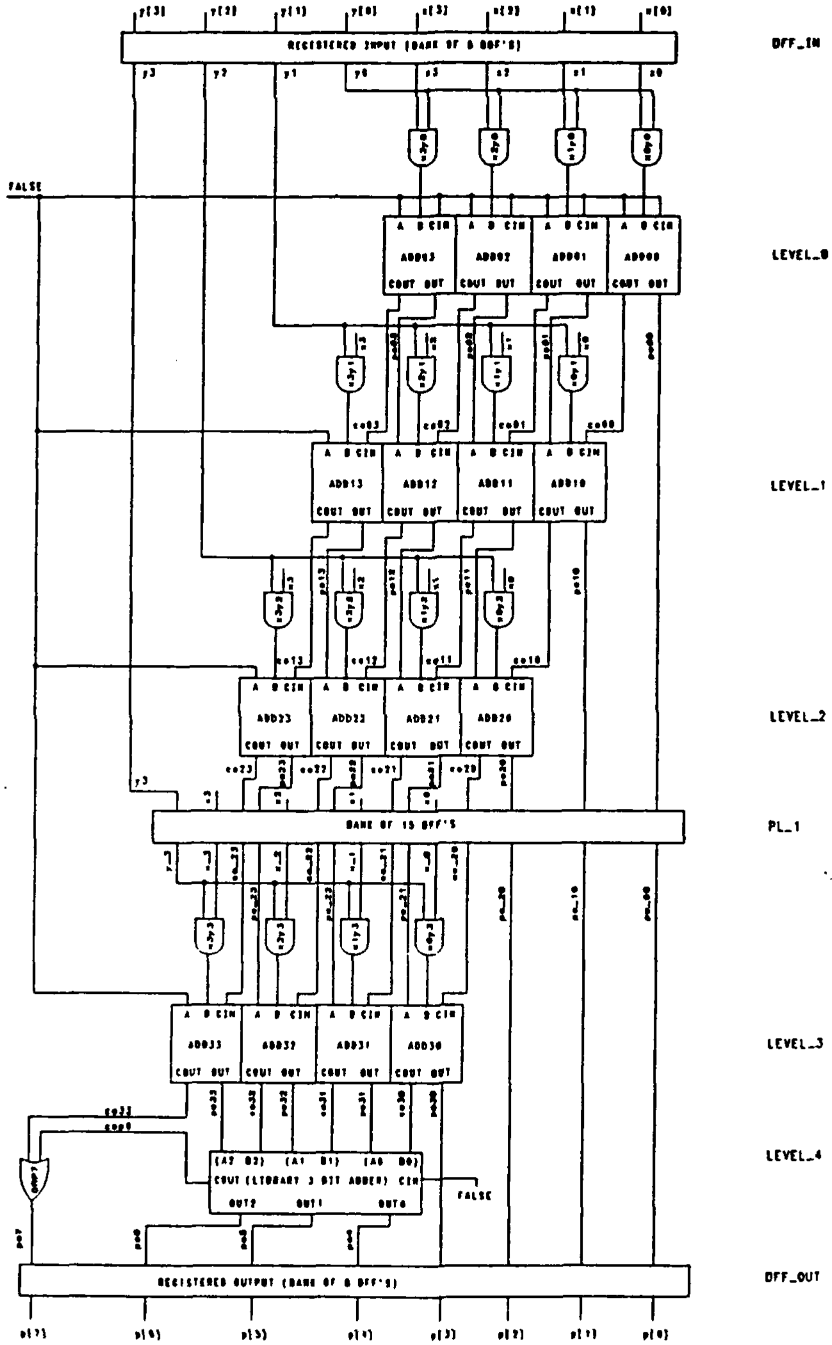


Figure 43 CAD Drawing of a 4-Bit Pipelined Multiplier Array (4bmmPL)

splitting up the library 3-bit adder into individual adder units as was previously done. The first stage requires approximately 17.6 ns to propagate the partial multiplication products while the second stage requires approximately 14.9 ns (32.5 ns - 17.6 ns). Here, one can see the limiting stage is comprised of level_0 thru level_2. In other words, the multiplier is limited to the pipelined stage with the longest delay. However, one must also include the delay of the D flip-flops in the overall timing calculation. The theoretical clock period (T) is determined from the sum of the longest pipelined stage delay plus the flip-flop delay and the setup time for the flip-flops. Here, the assumption is made that all stages in the pipeline receive the same clock pulse simultaneously. In reality, due to circuit lengths, loading, and driver circuits it is nearly impossible to guarantee that all stages of a pipelined circuit receive the same clock pulse at exactly the same time. From Table 2, and Figures 32 and 44, T is estimated at 23.1 ns [17.6 ns (slowest stage delay) + 4.0 ns (D flip-flop delay) + 1.5 ns (setup time)].

```
Netlist: [illegible]                                         Timing Analyzer
SETUP AND HOLD MODE
[illegible]                                  Corner: [illegible]
  Junction Temperature:75 degree C           Voltage:5.00V
  Phase 1: phase_a                           Phase 2: phase_b
 Included setup files: default setup file
```

INPUT SETUP AND HOLD TIMES (ns)

Input	Setup Time Ph1(f)	Setup Time Ph2(f)	Hold Time Ph1(f)	Hold Time Ph2(f)	
x[0]	---	1.5	---	-0.4	Fall
x[1]	---	1.5	---	-0.4	Fall
x[2]	---	1.5	---	-0.4	Fall
x[3]	---	1.5	---	-0.4	Fall
y[0]	---	1.5	---	-0.4	Fall
y[1]	---	1.5	---	-0.4	Fall
y[2]	---	1.5	---	-0.4	Fall
y[3]	---	1.5	---	-0.4	Fall

Figure 44 Input Setup and Hold Times for 4bmmPL

The corresponding clock frequency was estimated at approximately 43 MHz (1/T). The theoretical clock frequency for 4bmm2.RIRO was determined to be approximately 26 MHz (1/38 ns) [32.5 ns (delay for entire array) + 4.0 ns (D flip-flop delay) + 1.5 ns (setup time)]. 4bmmPL illustrates the increase in throughput when pipelining is employed. The GENESIL floorplan and layout for 4bmmPL are shown in Figures 45 and 46.

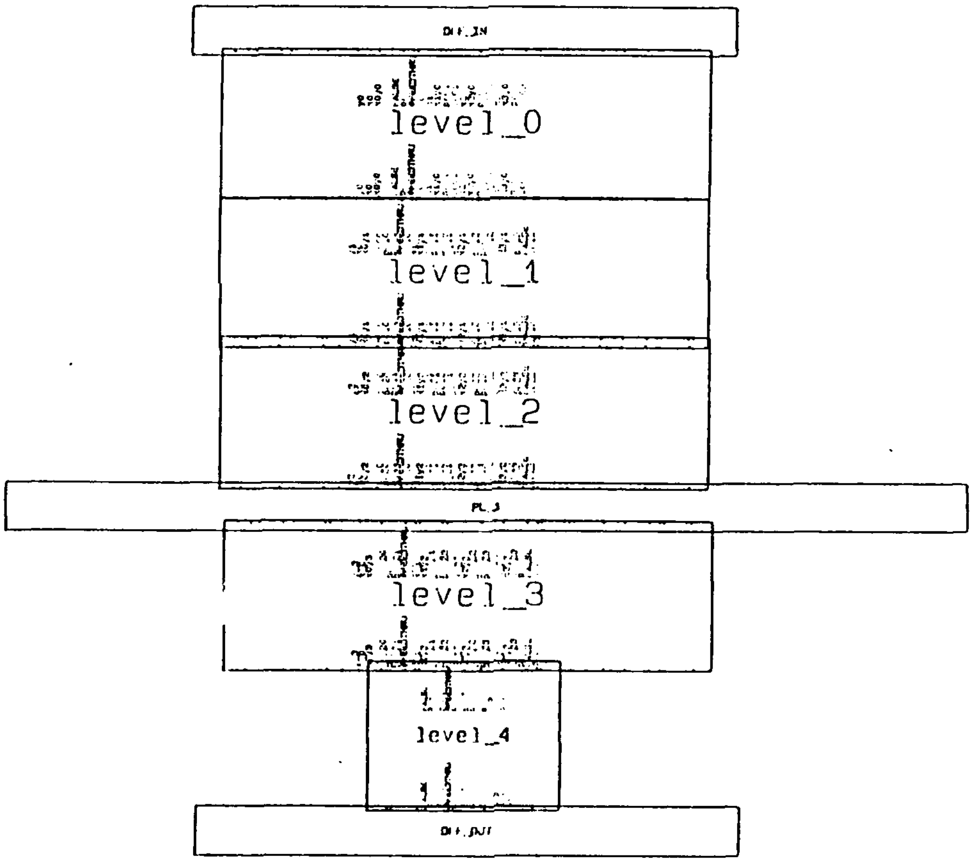


Figure 45 Floorplan for 4bmmPL

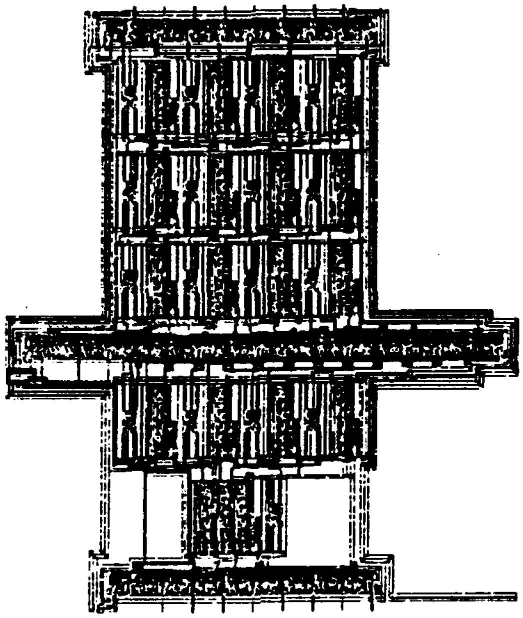


Figure 46 GENESIL Layout of 4bmmPL (4,455.45 mils^2)

Following the construction and functional verification of 4bmmPL, a timing analysis was performed to determine the accuracy of the predicted clock speed vs. the actual clock speed as determined by GENESIL. The option "clocks" was used to determined the worst case paths. From inspection of Figure 47, one can see that the worst case path was determined to be 24.6 ns or approximately 40 MHz. This indicates the predicted value was in error by approximately 7%. It is assumed that when the circuit is tested as a whole, greater accuracy is achievable due to simulation of the loading conditions, as well as circuit length delays.

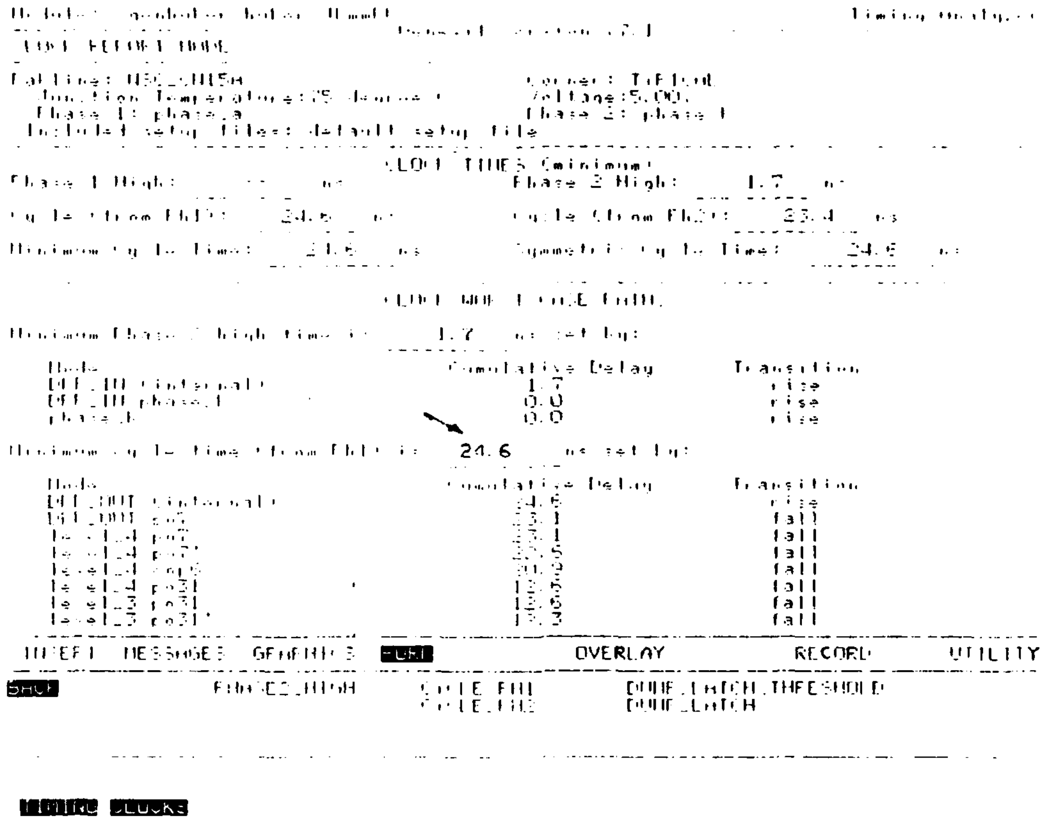


Figure 47 Clock Worst Case Paths for 4bmmPL

After a timing analysis was conducted, the orientation of the levels and registers were varied to determine if a smaller layout could be attained.

The first attempt at decreasing the layout of 4bmmPL was to use GENESIL's AUTO_PLACEMENT algorithm instead of manual PLACEMENT during the FLOORPLANNING process. The resulting floorplan is shown in Figure 48. It reveals a totally different perspective on arranging the Blocks which comprise 4bmmPL. One can see how the algorithm placed the pipeline register (PL_1) next to the input and output registers, DFF_IN and DFF_OUT respectively. The resulting GENESIL layout is shown is Figure 49. GENESIL's

AUTO_PLACEMENT algorithm was able to reduce the layout by approximately 28% by simply rearranging the Blocks during FLOORPLANNING.

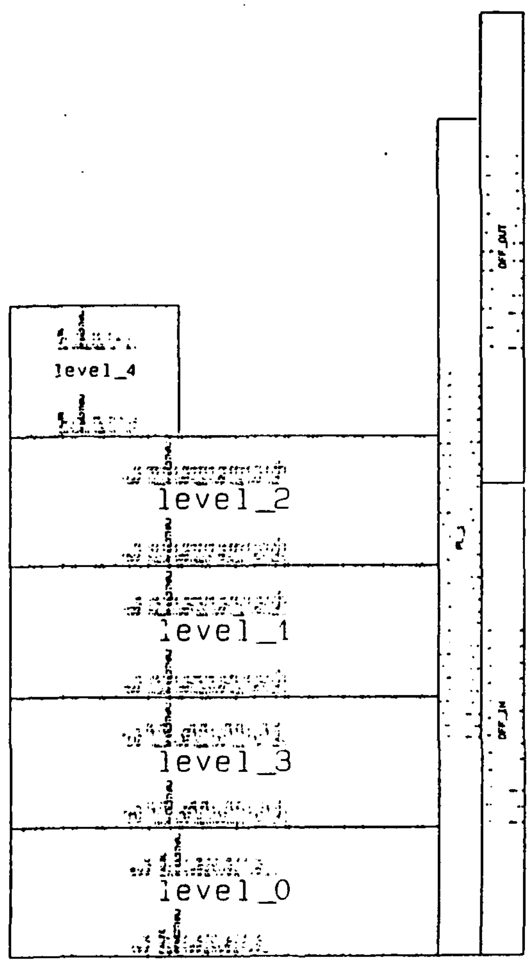


Figure 48 Floorplan from AUTO_PLACEMENT of 4bmmPL

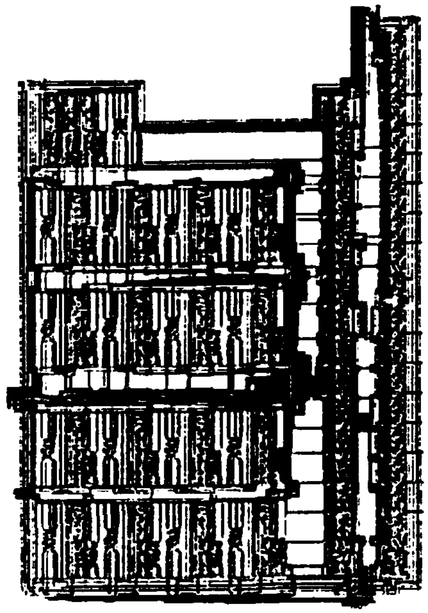


Figure 49 GENESIL Layout of 4bmmPL After AUTO_PLACEMENT (3,476.5 mils^2)

After observing the results of GENESIL's AUTO_PLACEMENT algorithm, the author decided to "challenge" GENESIL's algorithm by splitting PL_1 of Figure 43 in an attempt to further reduce the total area of 4bmmPl. The splitting was accomplished by using two banks of D flip-flops. One bank contained 8 flip-flops and the other 7. The two banks, labeled PL_1A and PL_1B, were manually placed at the sides of levels 1, 2, and 3 as illustrated in Figure 50. The resulting GENESIL layout is shown in Figure 51. Here, one can also see the difference between what is shown in the floorplan view and the final

GENESIL layout. This orientation did not result in a smaller total area than that achieved by GENESIL's AUTO_PLACEMENT algorithm; 3477.5 $mils^2$ versus 3850.7 $mils^2$.

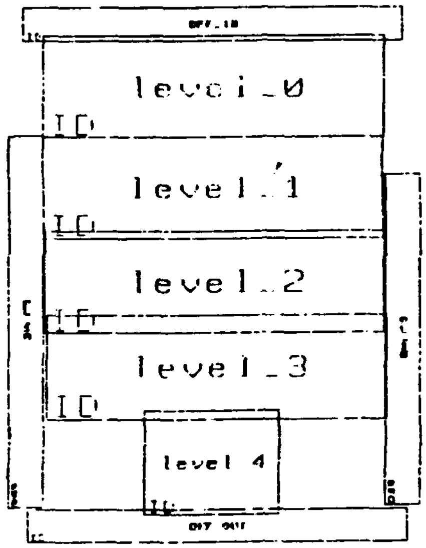


Figure 50 Floorplan of Split PL_1A and PL_1B of 4bmmPL

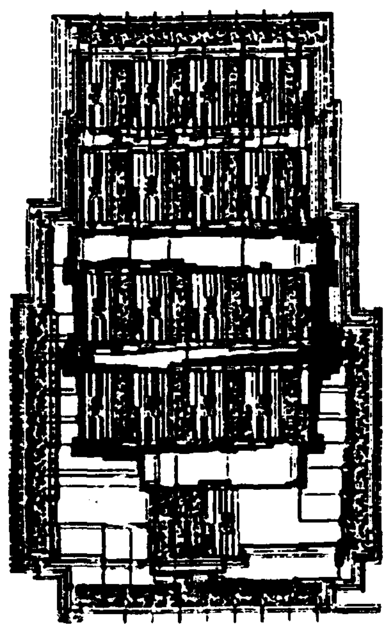


Figure 51 GENESIL Layout of Split PL_1A and PL_1B of 4bmmPL (3,850.72 $mils^2$)

A final attempt at reducing the area was accomplished by stacking PL_1A on top of PL_1B, and then positioning them between levels 2 and 3. AUTO_FUSION was then selected. The resulting layout is shown in Figure 52.

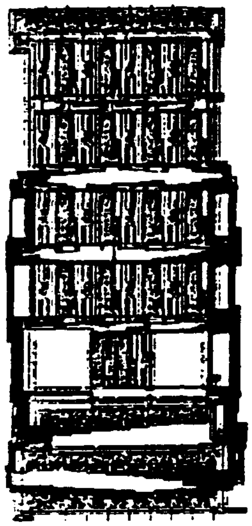


Figure 52 Stacking of PL_1A and PL_1B of Split 4bmmPL

A rather surprising result was observed. It appears that the AUTO_FUSION option "pushed" the two stacked registers below the final adders even though the were manually placed between levels 2 and 3. This orientation was not successful in reducing the total area as was AUTO_PLACEMENT. Therefore, one must conclude that GENESIL's AUTO_PLACEMENT algorithm is better able to place the individual Blocks of 4bmmPL to achieve a smaller total area. Even though it was demonstrated that the orientation in Figure 49 resulted in the smallest total area, it was decided to incorporate the orientation of Figure 46 into a Chip Module to better illustrate the concept of pipelining. Figure 53 shows the floorplan for the 4-bit multiplier Chip (4bmulti_chip) and its GENESIL layout is shown in Figure 54.

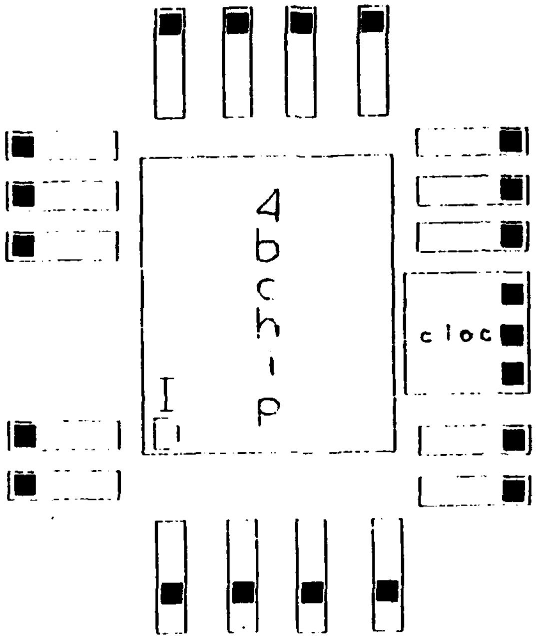


Figure 53 Floorplan of 4bmulti_chip

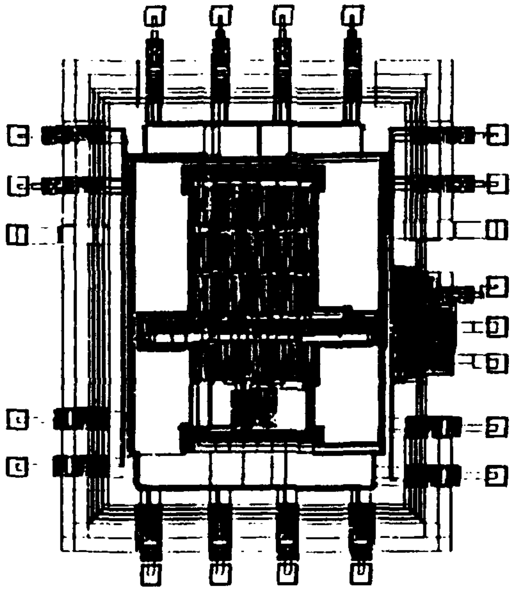


Figure 54 GENESIL Layout of 4bmulti_chip (19,806.15 mils^2)

Note that the Chip Module 4bmulti_chip is approximately 445% greater in total area than 4bmmPL.

C. DESIGN OF AN 8-BIT PIPELINED MULTIPLIER ARRAY

1. 8-Bit Multiplier Array

After the design of the 4-bit pipelined multiplier array was completed, efforts were directed towards developing the layout of an 8-bit pipelined multiplier. The same basic techniques used in the development of the 4-bit multiplier were applied.

A. Version 1

The first step was to extend the CAD drawing of Figure 22 to an 8-bit array. Figures 55 and 56 show the CAD drawing for an 8-bit parallel multiplier array (version 1 was labeled 8bmm.1). Note the final row of adders. Each final adder (FAP8-FAP14) is a 1-bit full adder. The carryout of each adder is rippled to the adjacent adder to the left. A generic level_*k*, comprised of 8 full adders and 8 AND gates, was employed to construct the array.

The AUTO_PLACEMENT algorithm was used during FLOOR-PLANNING in order to evaluate its placement of the blocks for the array. Figure 57 shows the results of GENESIL's AUTO_PLACEMENT algorithm for 8bmm.1. One can see a similarity to Figure 24. Note how the AUTO_PLACEMENT algorithm in both cases positioned the smallest block at the top of the array. Also, note in Figure 57 that the levels are not arranged in the order of "logic flow." Figure 58 shows the GENESIL layout for 8bmm.1 with a total area of 8157.5 mils^2. One can see a thickening of metal between level_2 and the other adder levels, as well as to the left of the array in both the upper and lower regions.

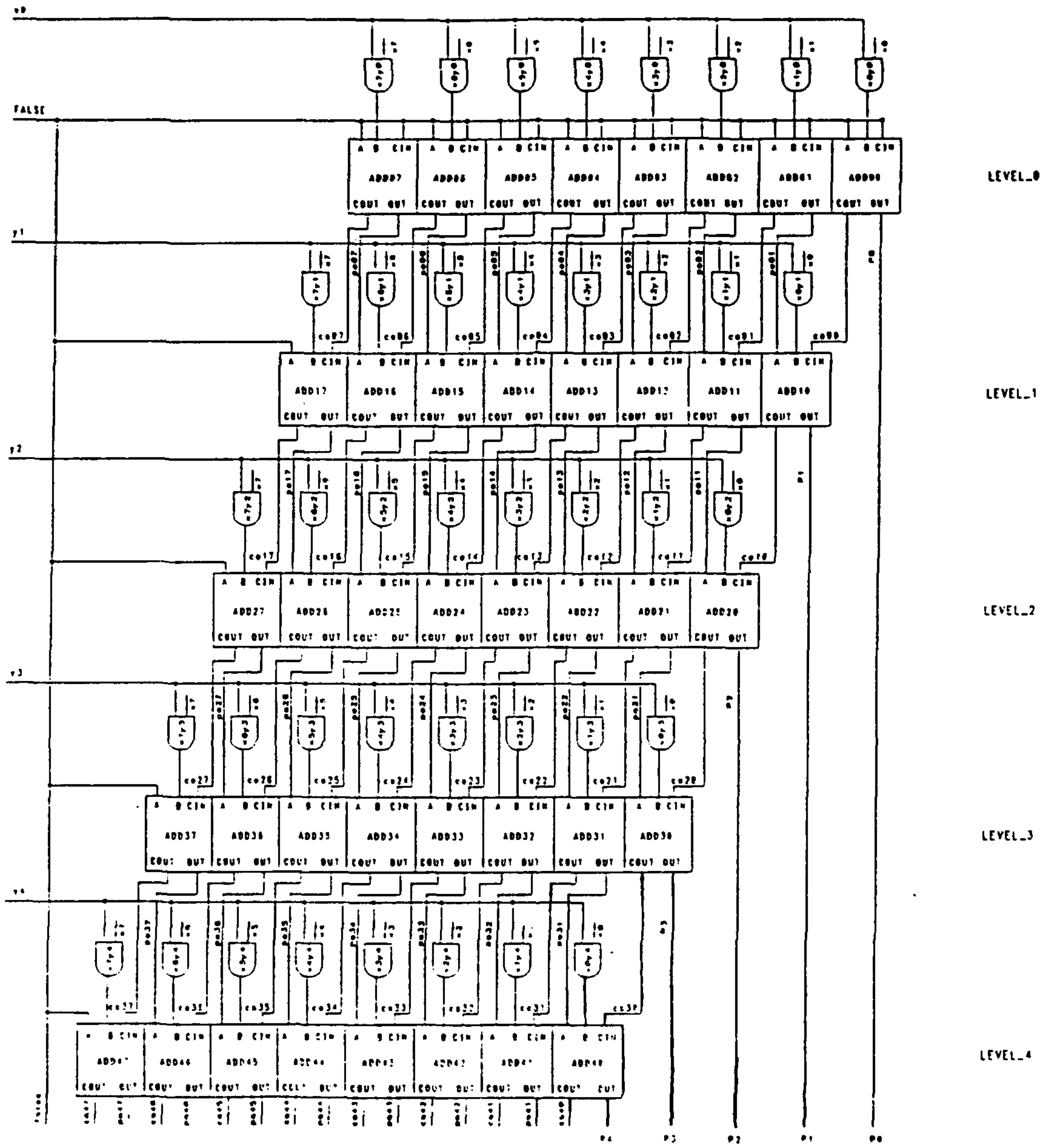


Figure 55 CAD Layout (Upper Half) for 8bmm.1

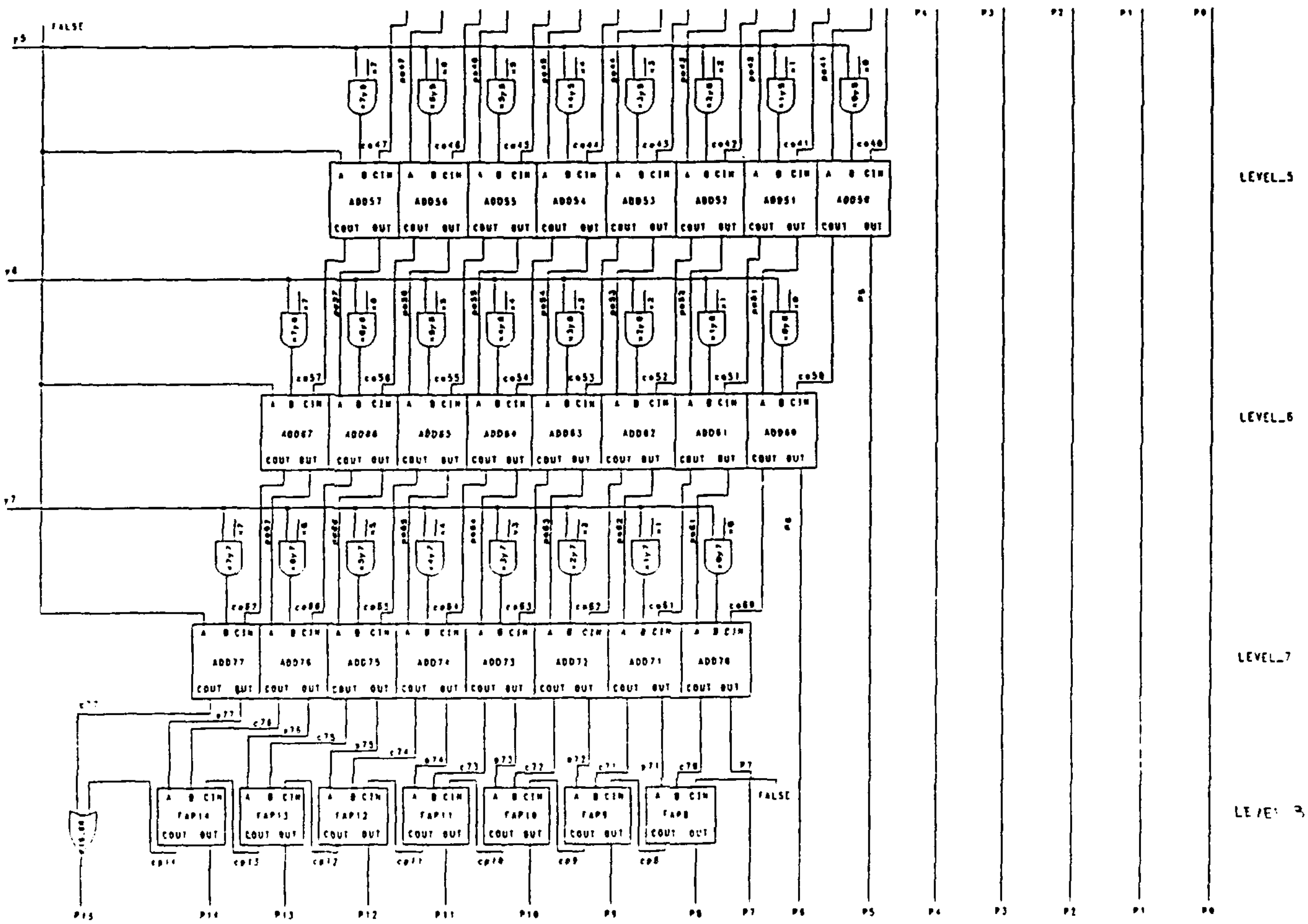


Figure 56 CAD Layout (Lower Half) for 8bmm.1

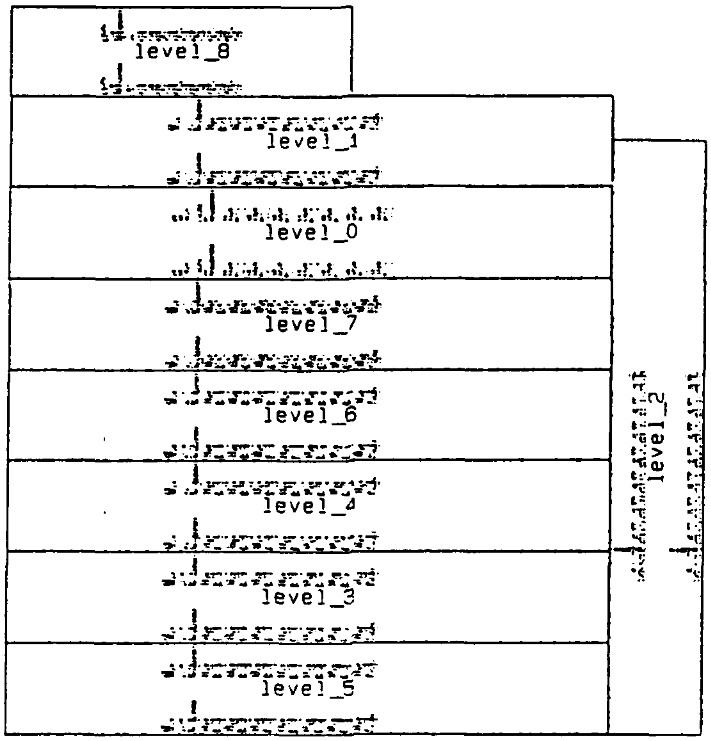


Figure 57 Floorplan for 8bmm.1

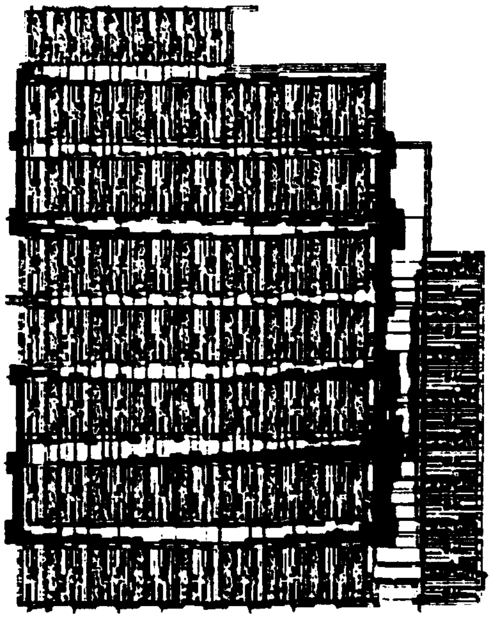


Figure 58 GENESIL Layout for 8bmm.1 (8,157.51 mils²)

Before further modifications to the array were made, the functionality was verified. Following the functional verification, a timing analysis was conducted and the results are shown in Figure 59.

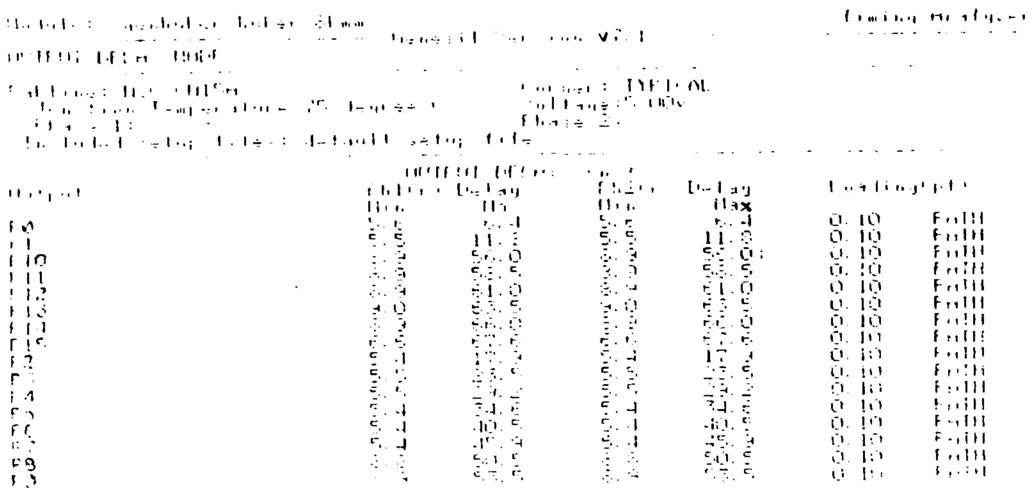


Figure 59 Timing Analysis for 8bmm.1

B. Version 2

Following the functional verification and timing analysis for 8bmm.1, the orientation of the ADDER/AND levels of the multiplier was changed to reflect the order of logic flow. The floorplan for this orientation (labeled 8bmm.2) is shown in Figure 60. Note the spacing between the levels of the floorplan. This was done for comparison with the next iteration to determine what effect spacing and overlap would have on the overall multiplier size. Figure 62 shows the resulting GENESIL layout. Comparing Figures 58 and 61, one can see the latter is a "cleaner" looking layout with minimal metal running throughout the array. The resulting area was calculated to be approximately 8474.23 mils^2 compared to 8157.51 mils^2 for 8bmm.1. This represents approximately a 4% increase in area. A timing analysis was also conducted to determine if this orientation resulted in a lower propagation delay for P15. The

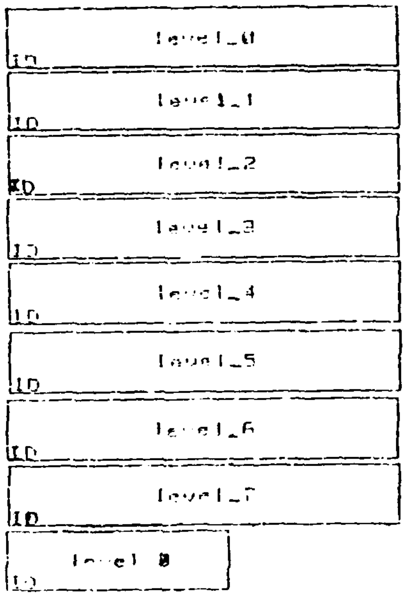


Figure 60 Floorplan for 8bmm.2

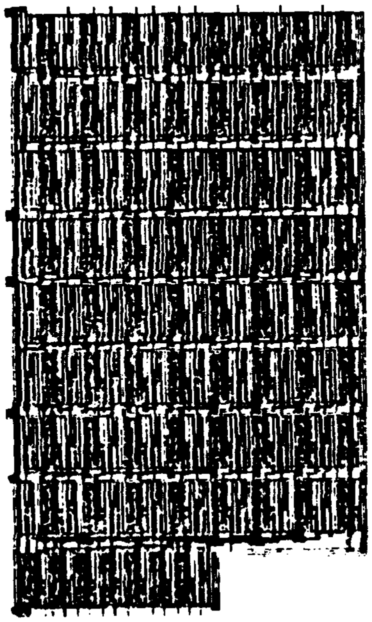


Figure 61 GENESIL Layout of 8bmm.2 (8,474.23 $mils^2$)

results of the timing analysis indicate that there was no significant difference in the propagation delay for P15 (52.3 ns vs 53.5 ns for 8bmm.2 and 8bmm.1, respectively).

C. Version 3

The next iteration (8bmm.3) was done specifically to determine if the multiplier area could be reduced if adjacent levels were slightly overlapped during FLOORPLANNING. Figure 62 shows how the individual layers were manually placed and overlapped during the FLOORPLANNING process. The resulting layout for 8bmm.3 was similar to Figure 61.

Figure 62 Floorplan for 8bmm3

The resulting area was calculated to be 8513.23 mils^2. This represents an increase of approximately 1% over 8bmm.2. This suggest that overlapped levels will be separated by a slightly greater amount than if they were adjoining each other.

D. Version 4

The next iteration (8bmm.4) was a modification to 8bmm.3 by replacing the final individual 1-bit adders with a 7-bit adder. As observed in 4bmm.2, it was expected that the propagation delay of the final product (here P15) would be reduced. Figure 63 shows this modification to level_8. The floorplan for 8bmm.4 was identical to 8bmm.3 (see Figure 62).

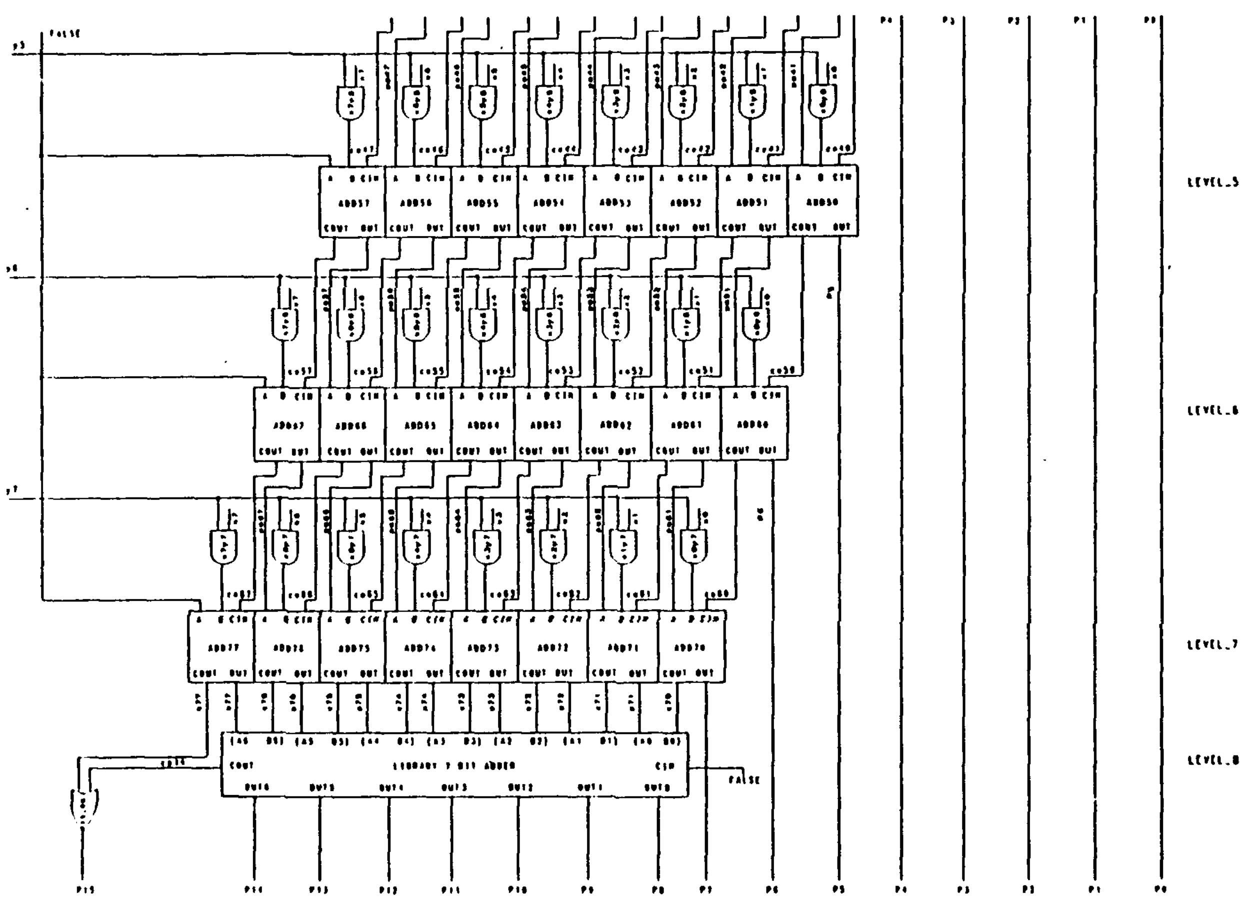


Figure 63 8bmm.4 (7-Bit Adder)

The resulting layout is shown in Figure 64. Close inspection of level_8 reveals a higher density for the 7-bit adder than for the individual adders of 8bmm.3. A timing analysis was performed on 8bmm.4 and the results are shown in Figure 65. As expected, the delay for P15 of 8bmm.4 was reduced by 6.5 ns (67.6 ns -

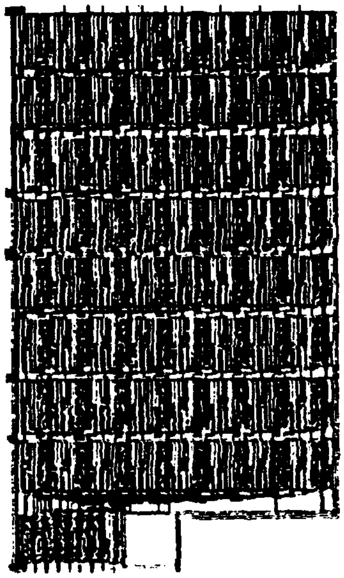


Figure 64 GENESIL Layout for 8bmm.4 (8,539.21 mils²)

Genesil Version 7.1 — Timing Analyzer

Voltage:5.00

Phase 2:

default setup file

OUTPUT DELAYS (ns)

Output	Phase 1 Delay Min	Phase 1 Delay Max	Phase 2 Delay Min	Phase 2 Delay Max	Loading(pF)	
P0	6.0	6.3	6.0	6.3	0.10	[illegible]
P1	6.8	12.1	6.8	12.1	0.10	[illegible]
P10	8.1	53.4	8.1	53.4	0.10	[illegible]
P11	8.3	54.7	8.8	54.7	0.10	[illegible]
P12	8.1	56.6	8.1	56.6	0.10	[illegible]
P13	8.6	[illegible]	8.6	57.9	0.10	[illegible]
P14	8.1	59.5	8.1	59.5	0.10	[illegible]
P15	6.6	61.1	6.6	61.1	0.10	[illegible]
P2	[illegible]	17.8	[illegible]	17.8	0.10	[illegible]
P3	5.7	23.1	5.7	23.1	0.10	[illegible]
P4	[illegible]	28.9	[illegible]	28.9	0.10	[illegible]
P5	5.6	34.3	5.6	34.3	0.10	[illegible]
P6	5.5	40.0	5.5	40.0	0.10	[illegible]
P7	5.5	45.3	5.5	45.3	0.10	[illegible]
P8	5.3	49.7	5.3	49.7	0.10	[illegible]
P9	[illegible]	51.5	6.3	51.5	0.10	[illegible]

Figure 65 Timing Analysis for 8bmm.4

61.1 ns) which represents an reduction of approximately 6% in propagation delay.

E. Version 5

The last iteration of this particular orientation centered the final row of adders directly below the last level of the array as in 4bmm.4. The layout (8bmm.5) is shown in Figure 66 which resulted in a reduction of approximately 2% in total area over that of 8bmm.4. Also, there was no change in the timing analysis; it was the same as for 8bmm.4 (Figure 65).

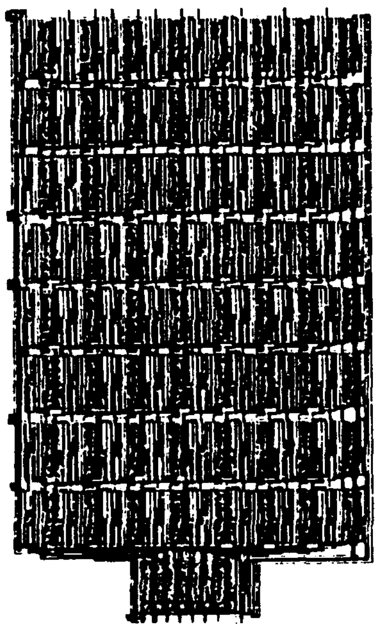


Figure 66 GENESIL Layout of 8bmm.5 (8,395.65 mils^2)

F. Version 6

The last version of the 8-bit multiplier (8BITMOD) array was constructed from four 4-bit multiplier array modules (see Figure 22). The floorplan for 8BITMOD is shown in Figure 67. Each 4-bit multiplier array module was attached to a common general module, as well as a single random logic Block containing the final adders. Although this particular orientation did not result in a reduction in total area, the design was very useful in learning how

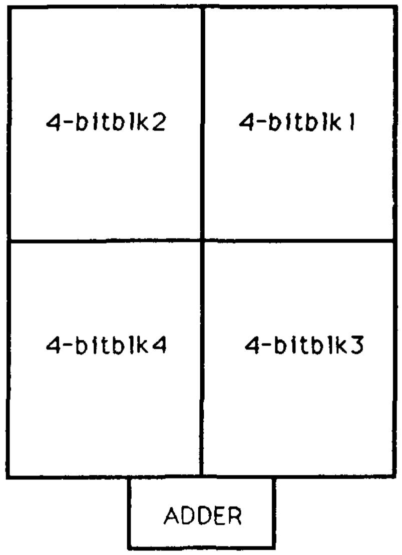


Figure 67 Floorplan for 8BITMOD

to use OBJECT_NETLIST and NET_NETLIST. 8BITMOD required extensive use of OBJECT_NETLIST when interconnecting the four individual modules, particularly, when routing signals across the module boundaries. For example, a signal can be identified inside a module as signal "x" but when the signal line leaves the module and is routed to another module, one can change its name to signal "y". This property was very useful and minimized the requirement to "customize" each individual 4-bit multiplier. The GENESIL layout for 8BITMOD is shown in Figure 68. The total area is approximately 8993.1 mils^2. This was the largest of the 8-bit parallel multiplier arrays.

Before starting the design of the pipelined version of the 8-bit parallel multiplier array, a decision had to be made regarding what orientation to implement. Based on size only, 8bmm.1 (Figure 58) would be favored because it had the smallest area. However, due to the size (width) of the D flip-flops required to pipeline the array, the orientation of 8bmm.5 (see Figure 66) was selected. The decision to implement the orientation of 8bmm.5 was also based on

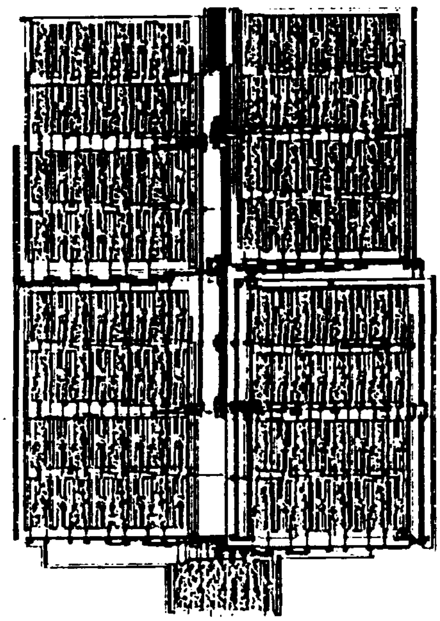


Figure 68 GENESIL Layout for *8BITMOD* (*8,993.1 mils²*)

the inherent symmetry of the array which would lend itself to simple horizontal cuts for inserting the pipeline registers.

2. 8-Bit Pipelined Multiplier Array

The first step in designing the pipelined 8-bit multiplier array was to inspect the timing analysis of 8bmm.5 to determine between what levels the pipelined registers should be inserted. Based on the output delays of 8bmm.5 listed in Table 3, the array was divided into four pipelined stages. The product out of the first stage (P2) was available after a 17.6 ns propagation delay and the outputs from the other stages were nearly a multiple of this delay.

TABLE 3 TIMING ANALYSIS FOR 8BMM.5

Product	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15
Delay (ns)	6.8	12.1	17.8	23.1	28.9	34.3	40.0	45.3	49.7	51.5	53.4	54.7	56.6	57.9	59.8	61.1

Table 3 suggest inserting registers between products P2/P3, P5/P6, and P9/P10 which will result in nearly equal delays for each stage. This corresponds to inserting registers between levels 2/3, 5/6, and P9/P10 of Figures 55 and 63. The insertion of registers between P9/P10 required a modification to the final row of adders in level_8. This modification (8bmm.5A) is shown in Figure 69 below. It was necessary to split the original 7-bit adder of 8bmm.5 into a 5-bit and 2-bit adder to accommodate the insertion of the final pipeline registers. A

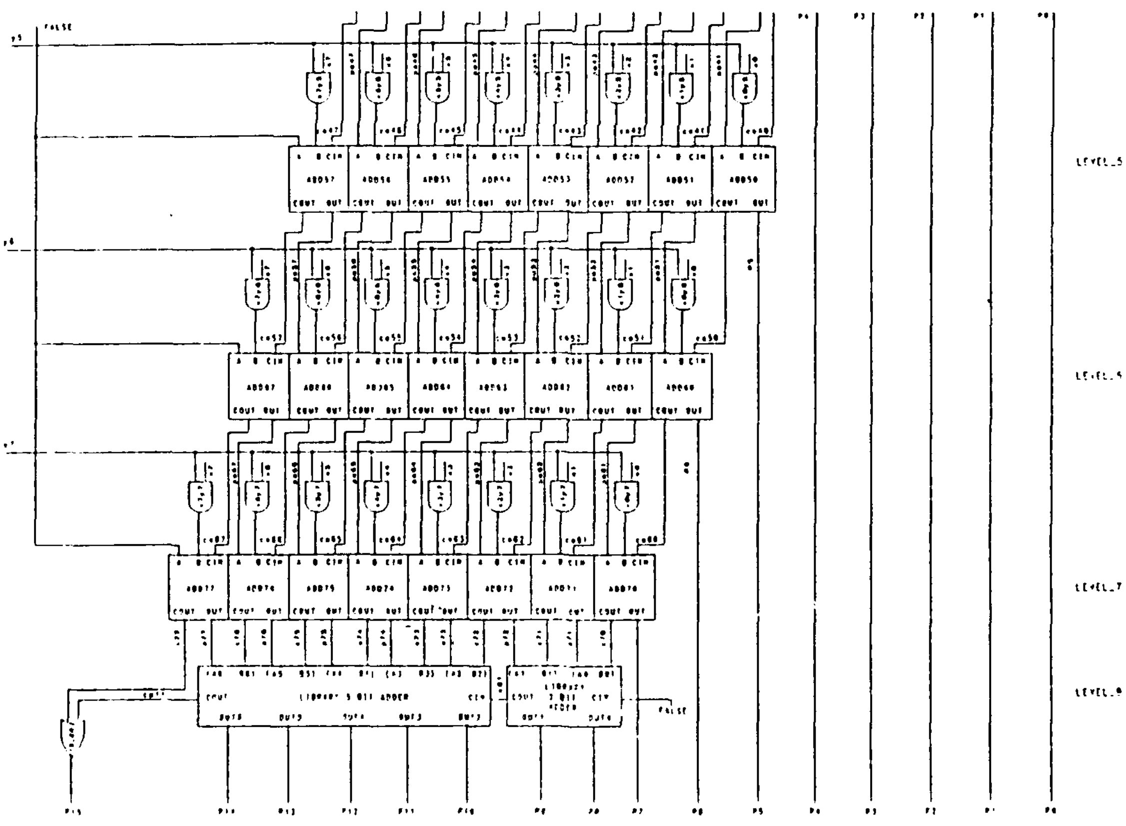


Figure 69 Modification to Level_8 (8bmm.5A)

timing analysis was conducted on 8bmm.5A and the results are shown in Figure 70 below.

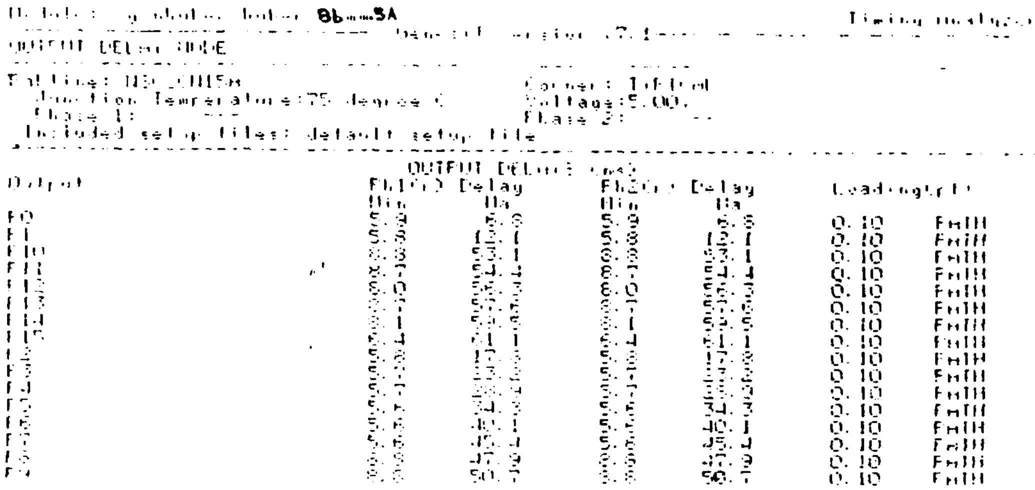


Figure 70 Timing Analysis for 8bmm.5A

The results show a 17.8 ns delay for the stage 1 (levels 0-2), a 16.5 ns delay for stage 2 (levels 3-5), a 16.4 ns delay for stage 3 (level_6 thru P9), and an 8.8 ns delay for stage 4, the final row of adders. This is summarized in Table 4 below.

TABLE 4 OUTPUT DELAYS FOR PIPELINED STAGES 1-4

STAGE	LEVELS	OUTPUT DELAYS (ns)
1	0-2	17.8
2	3-5	16.5
3	6-P9	16.4
4	P10-P15	8.8

Following the timing analysis, a CAD drawing depicting the pipelined 8-bit multiplier array (8bmmPL) was made. Figure 71 shows the upper third and Figure 72 shows the middle third of 8bmmPL. Figure 73 shows the lower third of this array.

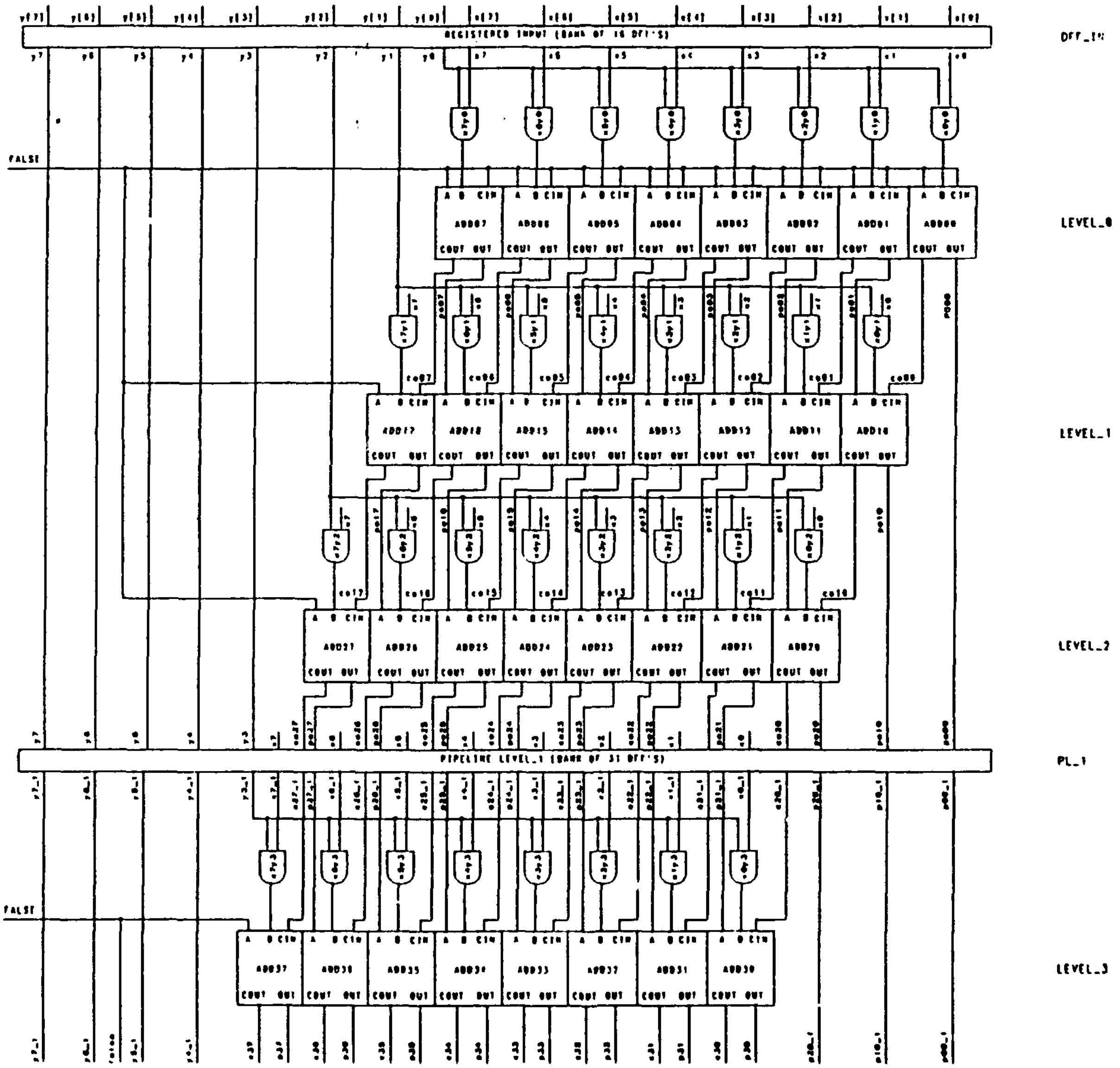


Figure 71 CAD Layout of 8bmmPL (Upper Third)

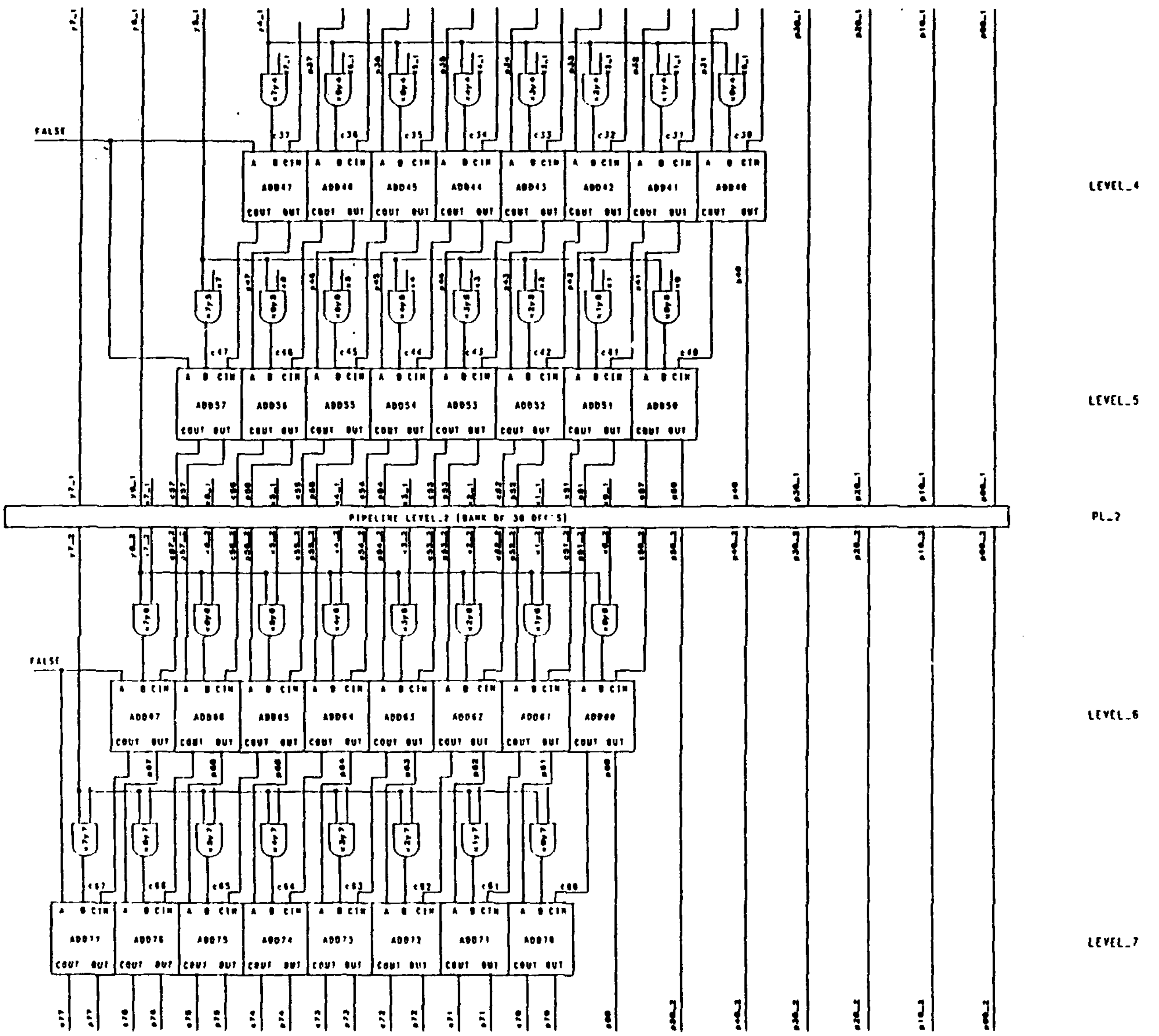


Figure 72 CAD Layout of 8bmmPL (Middle Third)

The basic signal naming scheme was modified, due to the presence of pipelined stages, by use of an underline character "_" to indicate signals which passed through pipelined stages.

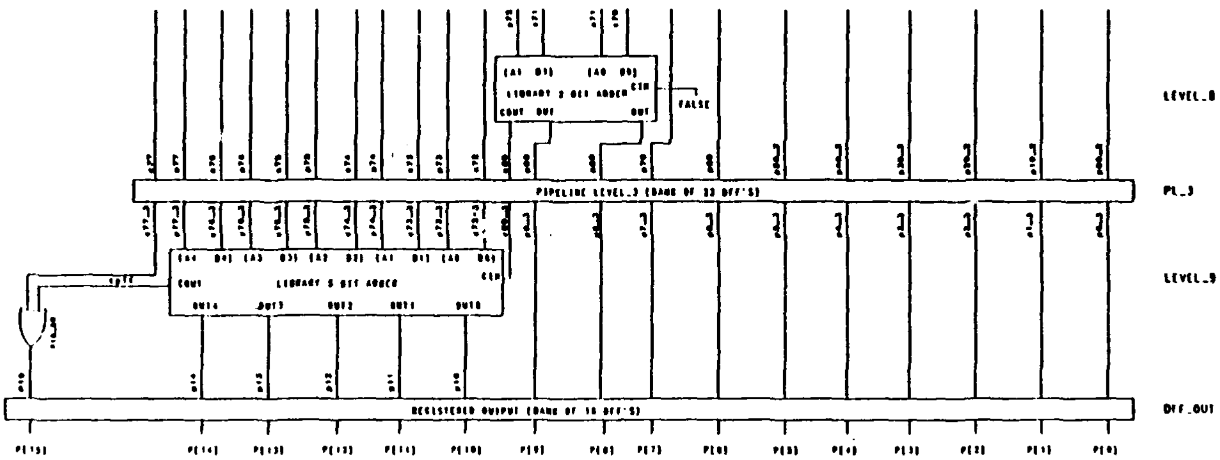


Figure 73 CAD Layout of 8bmmPL (Lower Third)

Note in Figure 73 how the first two adders are separated from the final row of adders in level_9. This resulted from the splitting of the original 7-bit adder in order to pipeline in four stages. The floorplan for the array is shown in Figure 74 and the GENESIL layout is shown in Figure 75. One can clearly see the individual levels and pipeline registers. However, one can also see unused spaced between the first two stages to the left and right of the array. One can also see the two adders, which produce P8 and P9, and the empty space surrounding them. Yet, overall, the structure clearly shows the logic flow of the array and demonstrates the physical concept of pipelining.

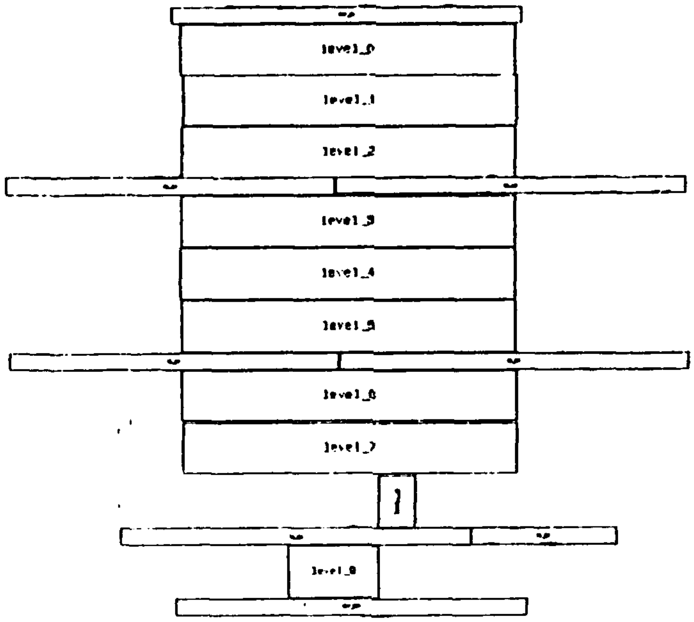


Figure 74 Floorplan for 8bmmPL

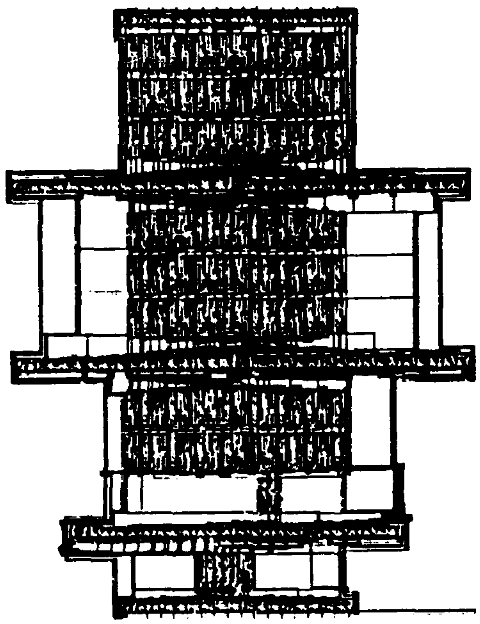


Figure 75 GENESIL Layout of 8bmmPL (20,000.67 mils^2)

Following the functional verification of 8bmmPL, a timing analysis was conducted to determine the worst case paths. The results are shown in Figure 76. The worst path was determined to be 26.7 ns which corresponds to clock rate of approximately 37.45 MHz.

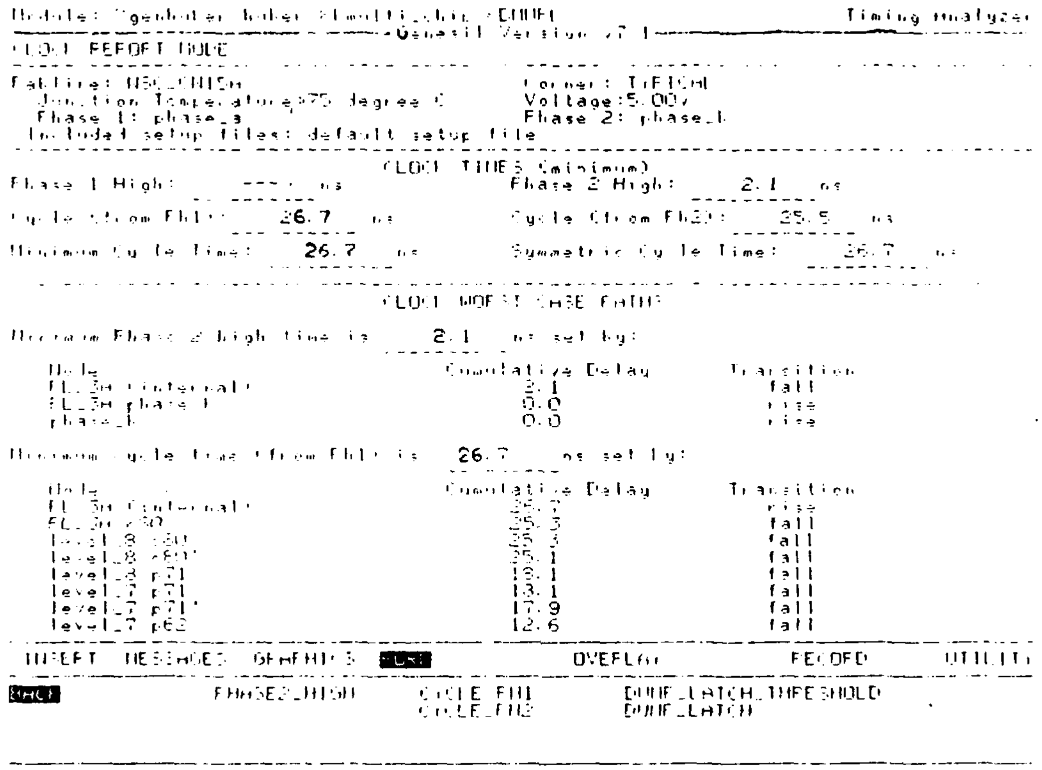


Figure 76 Worst Case Path for 8bmmPL

Finally, 8bmmPL was incorporated into a multiplier Chip (8bmulti_chip) which resulted in a total area of 44,488.41 mils^2. Note the Chip Module (8bmulti_chip) is approximately 222% greater in total area than 8bmmPL. Figure 77 shows the GENESIL layout for 8bmulti_chip.

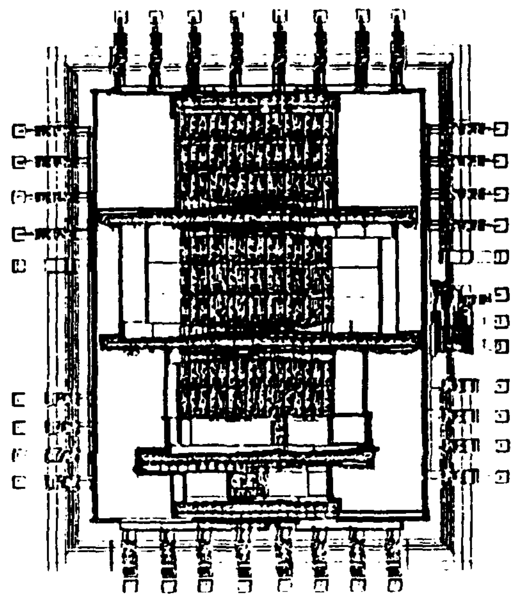


Figure 77 GENESIL Layout for 8bmulti_chip (44,488.41 mils^2)

3. 16-Bit Pipelined Multiplier Array

A 16-bit pipelined multiplier array, incorporating parallel multiplier cells, was not implemented in this study; however, from Figures 75 and 77 a projection of its core size (without PADS) was estimated to be 99,328 mils^2 (256 x 388), while its Chip size was estimated at 140,800 mils^2 (320 x 440). Figure 78 shows a Block level layout for this multiplier. Its operating speed was estimated at 38 MHz; the same as 8bmmPL.

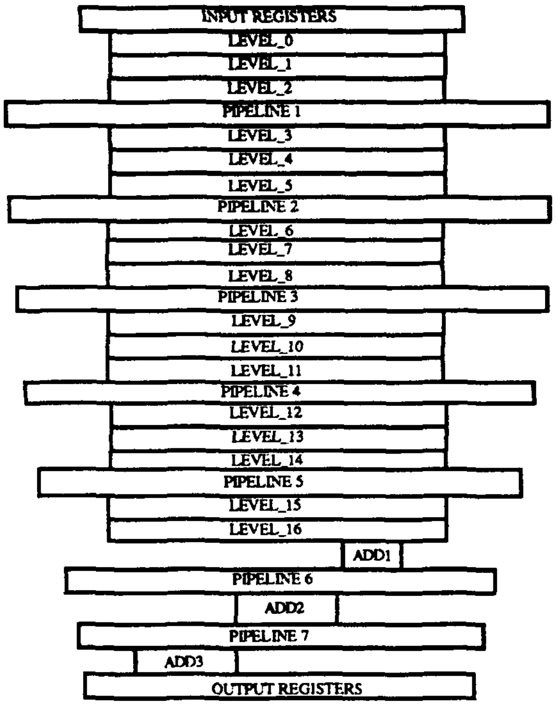


Figure 78 Block Level Layout of a 16-Bit Pipelined Multiplier Array

VI. LIMITATIONS OF THE SILICON COMPILER

It was a goal of this thesis to fully explore and probe the GENESIL Silicon Compiler system in order to determine its practical limits in parallel multiplier array design. During this course of study, two apparent limitations of the GSC system in parallel multiplier array design were discovered. They are:

- Component density.
- Vertical feedthrough.

The most significant limitation of the GSC system appears to be its inability to achieve high component density in parallel multiplier arrays of the type implemented in Chapter 5. Here, component density refers to the relative distance between levels of a parallel multiplier array, as well as between individual components comprising the array. It appears that high density is precluded because of the abutting of the power buses V_{DD} and V_{SS} of the individual components of the array. Figure 79 shows this abutment between adjacent components. Higher density might be achieved if the power buses of adjacent components were permitted to overlap. Additionally, the relative size (width) of the power buses appears to be a factor contributing to the separation between components.

The second limitation of the GSC appears to be its inability to establish vertical feedthrough between adjacent levels of ADDER/AND components in the parallel multiplier arrays in this study. As stated earlier, an attempt was made to increase the density of the arrays by collapsing the array vertically by moving the AND gate to the top of the ADDER and then rotating the two blocks clockwise 90°. After rotating the two blocks, a feedthrough Block was attached to each AND gate. This proved unsuccessful in passing the x_i from the AND gate of the upper level to the AND gate in the level below. Figure 80 shows just one of several attempts to establish vertical feedthrough.

Although the GSC system did not perform as desired in this study, it offers a viable alternative to the labor intensive, full custom, VLSI graphic layout tools in use today.

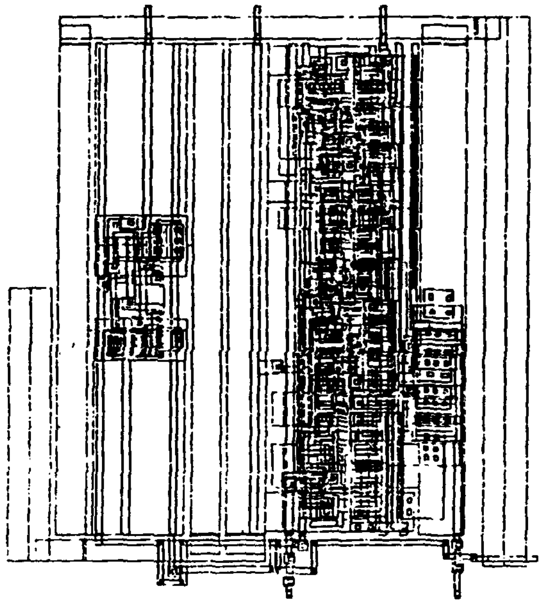


Figure 79 Abutment of ADDER/AND

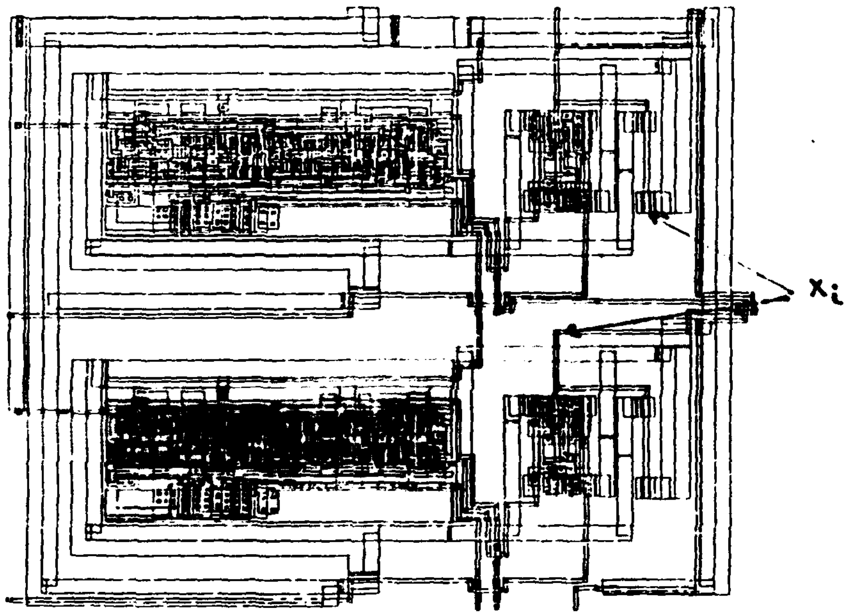


Figure 80 Vertical Feedthrough

VII. CONCLUSIONS

A. SUMMARY

The main goal of this thesis was to describe the design methodology and the process of employing the GENESIL Silicon Compiler (V7.1) in the layout of a pipelined multiplier, in 1.5 micron CMOS technology, using a parallel multiplier cell array. There was an additional goal of determining the practical limits of the GSC in parallel multiplier array design. Finally, there was the intent to produce a document with sufficient background material for those readers not well versed in digital design methodology in order that they might gain some understanding of the methods involved in the design of a pipelined parallel multiplier array.

The material in Chapter 2 provided a brief introduction to one particular silicon compiler, namely the GENESIL Silicon Compiler (GSC). Chapter 3 provided a review of the basic principles of digital multipliers, while Chapter 4 covered the basic concept and theory of pipelining. The design iterations of several pipelined parallel multiplier arrays, incorporating parallel multiplier cells, were presented in Chapter 5. Comments regarding the practical limits of the GSC system when implementing the parallel multiplier array designs of this study were presented in Chapter 6.

The results of this thesis indicate that a parallel multiplier array, incorporating parallel multiplier cells, can be successfully implemented in the GSC system. However, two practical limits of the GSC system precluded achieving the degree of high component density (smaller size) made possible by full custom manual/CAD design methods using graphic layout tools.

B. RECOMMENDATIONS

The author makes the following recommendations:

- Install version 8.0 of the GENESIL Silicon Compiler at the Naval Postgraduate School as soon as possible.
- Explore version 8.0 fully to determine its capability to establish vertical feedthrough. If successful, incorporate this feature into future parallel multiplier array designs for comparison with full custom manual/CAD designs using graphic layout tools.
- Investigate ways to reduce the CPU loading on the VAX system during normal working hcurs in order to enhance the performance of the GSC system.
- Allow for 3-4 months in learning to use the GSC. Preferably one should also attend the one week training course offered by Silicon Compiler System Corporation of San Jose, California.
- Incorporate the GSC system into, and make it a regular part of, a course of instruction at the Naval Postgraduate School.

LIST OF REFERENCES

1. Lee, J. Y., Garvin, H. L., and Slayman, C. W., "A High-Speed High-Density Silicon 8x8-bit Parallel Multiplier," *IEEE Journal of Solid-State Circuits,* Vol. SC-22, No. 1, pp. 35-39, February 1987.

2. Weste, N., and Eshraghian, K., *Principles of CMOS VLSI Design, A Systems Perspective*, Addison-Wesley, 1985.

3. Harata, Y., Nakamura, Y., Nagase, H., Takigawa, M., and Takagi, N., "A High-Speed Multiplier Using a Redundant Binary Adder Tree," *IEEE Journal of Solid-State Circuits*, Vol. SC-22, No. 1, pp. 28-33, February 1987.

4. Lu, F., and Samueli, H., "A Bit-Level Pipelined Implementation of a CMOS Multiplier-Accumulator using a New Pipelined Full-Adder Cell Design," *Proc. of 8th Annual International Phoenix Conf. on Computers and Communication*, pp. 45-65, IEEE Comput. Soc. Press, Washington, DC, USA, Cat. No. 89CH2713-6.

5. Evans, A. J., Mullen, J. D., and Smith, D. H., *Basic Electronic Technology*, Howard W. Sams & Company, 1987.

6. Hallin, T. G., and Flynn M. J., "Pipelining of Arithmetic Functions," *IEEE Transactions on Computers*, Vol. C-21, No. 8, pp. 880-886, August 1972.

7. Baugh, C. R., and Wooley, B. A., "A Two's Complement Parallel Array Multiplication Algorithm," *IEEE Transactions on Computers*, Vol. C-22, No. 12, pp. 1045-1047, December 1973.

8. Settle, R. H., "Design Methodology Using the GENESIL Silicon Compiler," Master's thesis, Naval Postgraduate School, Monterey, CA, September 1988.

9. *GENESIL System, System Description Users Manual*, Silicon Compiler System Corp., San Jose, CA, September 1987.

10. *GENESIL System, Timing Analysis*, Silicon Compiler System Corp., San Jose, CA, July 1987.

12. Davidson, J. C., "Implementation of a Design for Testability Strategy Using the GENESIL Silicon Compiler," Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1989.

13. Flynn, M. J., and Waser, S., *Introduction to Arithmetic for Digital Systems Designers*, Holt, Rinehart, and Winston, 1982.

14. Kogge, P. M., *The Architecture of Pipelined Computers*, Hemisphere Publishing, 1981.

15. Stuart, D., "VLSI Designs for Pipelined FFT Processors," Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1990.

BIBLIOGRAPHY

Baer, J., *Computer Systems Architecture*, Computer Science Press, 1980.

Carlson, D. J., "Application of a Silicon Compiler to VLSI Design of Digital Pipelined Carry Look Ahead Adder," Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1983.

Conradi, J. R., and Hauenstein, B. R., "VLSI Design of a Very Fast Pipelined Carry Look Ahead Adder," Master's Thesis, Naval Postgraduate School, Monterey, CA, September 1983.

Habibi, A., and Wintz, P. A.,"Fast Multipliers", *IEEE Transactions on Computers,* Vol. C-19, No. 2, pp. 153-157, August 1972.

Hayes, J. P., *Computer Architecture and Organization*, McGraw-Hill, 1979.

Hwang, K., *Computer Arithmetic, Principles, Architecture, and Design*, John Wiley & Sons, 1979.

Jangsri, V., "Infinite Impulse Response Notch Filter," Master's Thesis, Naval Postgraduate School, Monterey, CA, December 1988.

Kuck, D. J., *The Structure of Computers and Computations,* Vol. 1, John Wiley & Sons, 1978.

Mano, M. M., *Digital Design*, Prentice-Hall, 1984.

Simchik, R. J., "VLSI Design Of a Sixteen Bit Pipelined Multiplier Using Three Micron NMOS Technology," Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1985.

Stone, H. S., *Introduction to Computer Architecture*, Science Research Associates, 1980.

Taub, H., *Digital Circuits and Microprocessors*, McGraw-Hill, 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
4.	Curricular Officer, Code 39 Naval Postgraduate School Monterey, California 93943-5000	1
5.	Prof. Herschel H. Loomis, Jr., Code EC/LM Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	7
6.	Prof. Chyan Yang, Code EC/YA Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	5
7.	Commander, Naval Research Laboratory ATTN: LCDR Ronald S. Huber, Code 9120 4555 Overlook Ave., S.W. Washington, DC 20375	2